VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Nádstavba pro centralizovaný management IPv6 sítě

Centralized Management for IPv6 Network

2012 Jan Michálek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Jan Michálek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Nádstavba pro centralizovaný management IPv6 sítě
Centralized Management for IPv6 Network

Zásady pro vypracování:

Cílem práce je implementace systému, který konfiguračně zjednoduší centrální správu počítačové sítě menšího rozsahu, provozující protokol IPv6. Důraz při realizaci bude kladem zejména na uživatelskou přívětivost a snadnou konfigurovatelnost služeb prostřednictvím jednoho centrálního rozhraní. Systém bude umožňovat konfiguraci služeb jako je DNS, DHCPv6, služeb poskytovaných serverem RADIUS a s tím související správu uživatelských účtů. Dále bude umožňovat konfiguraci vhodné formy směrování jak na platformě zařízení společnosti Cisco tak na softwarových řešeních nad OS Linux.

1. Seznamte se s problematikou nasazení IPv6 v prostředí směrované sítě menšího rozsahu. Zaměřte se na nezbytné služby umožňující komfortní a přitom řízený přístup uživatelů k síti.
2. Prostudujte existující řešení, která se zabývají podobnou problematikou.
3. Navrhněte řešení umožňující snadnou centrální správu v prostředí menší počítačové sítě.
4. Na vhodně zvolené volně dostupné platformě váš návrh implementujte.
5. Vytvořené řešení otestujte a zhodnoťte dosažené cíle.

Seznam doporučené odborné literatury:

SMITH, Roderick W. Advanced Linux networking.: Addison-Wesley, 2002. 752 s. ISBN 9780201774238.
ODOM, Wendell; GEIER, Jim ; MEHTA, Naren. CCIE Routing and Switching Official Exam Certification Guide, 2nd Edition. USA : Cisco Press, 2006. 1104 s. ISBN 978-1-58720-141-7.
SCHRODER, Carla. Linux Networking Cookbook. USA : O Reilly Media, Inc., 2007. 640 s. ISBN 978-0-596-10248-7.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Martin Milata**

Datum zadání: 18.11.2011
Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.*

V Ostravě 22. dubna 2012 .....

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 22. dubna 2012 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli a konzultovali se mnou řešení, protože bez nich by tato práce nevznikla. Hlavně bych chtěl poděkovat vedoucímu této diplomové práce, Ing. Martinu Milatovi, jehož hluboké odborné znalosti a zájem o obor mi byly silnou oporou při jakémkoliv problému spojeném s vypracováním práce.

Abstrakt

Tato diplomová práce se zabývá problémem centrální konfigurace IPv6 sítí. Přestože neustále přicházejí zprávy o docházejícím adresním prostoru pro starší protokol verze 4, internetový protokol verze 6 se prosazuje velice pomalu. Mým předpokladem je, že přestože velké množství poskytovatelů internetového připojení již podporu tohoto protokolu nabízí, administrátoři menších počítačových sítí se zdráhají protokol nasadit. Důvodem může být, že nyní mají již zavedený fungující systém pro IPv4 a nová verze protokolu by znamenala nový software, nové konfigurace a navíc by administrace každého zařízení vyžadovala úpravy na dvou místech - pro každou verzi zvlášť. Navrhl a implementoval jsem proto software, který umí správu centralizovat a z jednoho místa spravovat záznamy pro DNS, DHCP a RADIUS na platformě Linux, přičemž nezáleží na verzi IP protokolu.

Klíčová slova: IPv4; IPv6; DHCP; DNS; RADIUS; Linux; Ubuntu; Java; NetBeans; JSF; Tomcat; Zebra; Quagga

Abstract

This Master's Thesis is focused on problem with centralized configuration of IPv6 networks. Although there are lots of messages about IPv4 address exhaustion coming, IPv6 succeeds very slowly. My assumption is that administrators of smaller computer networks are hesitating with implementation, although many Internet service providers already provides IPv6 support. One of the reasons may be the aversion to recreate existing networks with new software, new configuration and also any change made on device has to be configured twice - for every protocol version separately. Therefore I have designed and implemented software solution, which supports centralized management from single point for DNS, DHCP and RADIUS servers on Linux platform and version of Internet protocol does not matter.

Keywords: IPv4; IPv6; DHCP; DNS; RADIUS; Linux; Ubuntu; Java; NetBeans; JSF; Tomcat; Zebra; Quagga

Seznam použitých zkratek a symbolů

CGA	–	Common Gateway Interface
CIDR	–	Classless Inter-Domain Routing
DNS	–	Domain Name System
DHCP	–	Dynamic Host Configurarion Protocol
IETF	–	Internet Engineering Task Force
IOS	–	Internetwork Operating System
IP	–	Internet Protocol
IPv4	–	Internet Protocol version 4
IPv6	–	Internet Protocol version 6
ISP	–	Internet Service Provider
OSPF	–	Open Shortest Path First
POJO	–	Plain Old Java Object
RDBMS	–	Relational Database Management System
SQL	–	Structured Query Language
NAS	–	Network Access Server
NAT	–	Network Address Translation
OO	–	Obect Oriented
OOP	–	Obect Oriented Programming
WWW	–	World Wide Web

Obsah

1 Úvod

Cílem mé diplomové práce bylo navrhnout aplikaci pro snadnou správu počítačové sítě pro protokol IPv6. V současnosti se o IPv6 hodně mluví v souvislosti s vyčerpáním IPv4 rozsahů a už před několika lety se situace považovala za kritickou, nicméně společnosti se do jejího nasazení příliš dychtivě neženou. Důvodem může být zdánlivá složitost protokolu, kdy jsou uživatelé a někdy i administrátoři zvyklí vídat pouze IPv4 adresy a nová adresa v neznámé notaci je pro ně něco magického a proto nechtějí zavádět změny, dokud nebude vyhnutí. Situace je ovšem kritická už nyní, protože někteří regionální poskytovatelé již nemají volné bloky IPv4 adres na rozdělování. Mnoho odpůrců této technologie argumentuje tím, že se přináší větší složitost do stávajících sítí a místo jedné topologie je nutné spravovat dvě logické, každou pomocí jiných nástrojů a právě to byl popud pro vytvoření této práce, která by měla tuto činnost zjednodušit.

V zadání práce je uvedena centrální konfigurace IPv6 sítě. Během zpracovávání tématu jsem ovšem brzy zjistil, že by bylo pro administrátora takové sítě mnohem pohodlnější konfigurovat v jednom rozhraní IPv4 i IPv6 síť. Pokusil jsem se tedy analyzovat v současnosti nejpoužívanější software pro práci s IPv4 a najít jeho alternativu i pro IPv6. V několika případech bylo příjemné zjištění, že stávající software je stejně dobře použitelný pro obě verze protokolu (jednalo se např. o DNS server Bind9 nebo implementaci směrovacích protokolů jménem Zebra). U jiných součástí běžné sítě již vyvstaly problémy, ale jak je vidět v této práci, dokázal jsem se nimi vypořádat.

Práce je rozvržena na celkem 13 kapitol. Kapitoly 2 a 3 obsahují obecný úvod do problematiky počítačových sítí. Je zde rozebrána struktura IP adres a hlavní rozdíly mezi dvěma protokoly. Protože kvalitní literatury na toto téma je v současnosti dostupné velké množství a navíc mnohdy zdarma, nevěnoval jsem se popisu IP protokolů více než je nutné k uvedení do problematiky. V kapitole 4 jsem se pokusil shrnout obecné požadavky na aplikaci a operační systém, na kterém by měla být spouštěna. Zaměřil jsem se hlavně na snadnou použitelnost a současně robustnost.

Kapitoly 5, 6, 7 jsou věnovány popisu služeb běžně využívaných v dnešních sítích a každá kapitola současně obsahuje způsob, jakým jsem se vypořádal s implementací služby do aplikace. Pokusil jsem se vkládat do tohoto textu pouze části kódu a výpisy konfigurací, které jsou nutné k pochopení daného problému. Celou aplikaci, včetně všech konfiguračních souborů použitých při testování, je poté možné nalézt v příloze, která je součástí této práce. V části 8 lze nalézt způsob, jakým jsou šířeny směrovací informace a jak nakonfigurovat zařízení v síti, aby byla schopna se systémem komunikovat.

Kapitoly 9, 10 a 11 jsou věnovány aplikaci samotné. Jsou zde uvedeny technologie použité při implementaci, databázová struktura stojící na pozadí celé aplikace a také postup instalace a způsob práce s aplikačním serverem, jež aplikace využívá.

V kapitole 12 pak seznamím čtenáře s výsledným rozhraním a v závěrečné kapitole 13 zhodnotím dosažené výsledky

2 Počítačová síť

Počítačová síť (anglicky computer network) je souhrnné označení pro technické prostředky, které realizují spojení a výměnu informací mezi počítači. Umožňují tedy uživatelům komunikaci podle určitých pravidel, za účelem sdílení využívání společných zdrojů nebo výměny zpráv. Historie sítí sahá až do 60. let 20. století, kdy začaly první pokusy s komunikací počítačů. V průběhu vývoje byla vyvinuta celá řada síťových technologií. V poslední době jsou všechny sítě postupně spojovány do globální celosvětové sítě Internet, která používá sadu protokolů TCP/IP.[26] . V této práci se zaměřím výhradně na sítě postavené na sadě protokolů IP a to ve verzích 4 a 6. Proto pokud bude dále v práci zmiňována síť, síťový prvek apod., budou těmito pojmy myšleny pouze sítě TCP/IP postavené na protokolech IPv4 a IPv6.

2.1 Potřeby uživatele sítě

Základní smysl počítačové sítě je komunikace jednoho uživatele s ostatními. V síti se často používá analogie s doručováním klasické pošty, kterou také využiji pro popsání základních pojmů. Aby mohl uživatel komunikovat, musí mít k dispozici následující informace:

2.1.1 Adresa

Uživatel, který chce přijímat komunikaci potřebuje svojí unikátní **adresu**. Na odchozí zásilky bude uveden jako odesilatel, na příchozí jako adresát. V IP sítích je představována IP adresou.

Je možné, že na jedné adrese bydlí více uživatelů. Ti si poté domluví systém, jakým si budou příchozí poštu rozdělovat, kdy např. každý bude mít vlastní schránku unvitř budovy. V IP sítích je tato služba pojmenovaná NAT a každý uživatel má svojí IP adresu unikátní pouze v rámci prostoru skrytého za touto službou. Co kdyby ovšem každý uživatel chtěl svojí vlastní unikátní adresu? Nebo dokonce několik? Ve světě doručování pošty to je celkem nereálné, ve světě počítačů to již zdaleka není nemožné. Bohužel když autoři před padesáti lety systém navrhovali, nečekali tak masové rozšíření a tak mu dali k dispozici „pouze“ cca 4 miliardy celosvětově unikátních adres. To byl jeden z důvodů příchodu nového IP protokolu IPv6, který by měl jednou nahradit stávající IPv4. Více o rozdílech mezi protokoly v kapitole 3.

Další otázkou je, jak nově příchozí uživatel získá adresu? Nejspíše zažádá úřad v daném místě o její přidělení. V IP sítí může zajít za administrátorem, který mu adresu ručně nastaví. Také ale může využít pohodlnější možnost, kdy uživatel (a hlavně administrátor) nemusí nikam chodit, pouze pošle žádost na broadcast adresu a pokud je k dispozici DHCP server, který je ochoten mu adresu přidělit, dostane jí automaticky. Podrobnější popis tohoto systému lze nalézt v kapitole 5. Současně jako bonus může obdržet i adresy DNS serverů a výchozí brány, viz dále. Je zde také ještě jedna možná alternativa, kterou přinesl protokol IPv6 a tou je autokonfigurace. V analogii by ji bylo možné popsat zhruba tak, že místní poštovní úřad pravidelně rozesílá do všech schránek letáky s dostupným rozsahem adres a uživatel si za tuto adresu připojí svůj jedinečný identifikátor (např. rodné číslo) a rovnou jí začně používat. Více o tomto systému je uvedeno v kapitole 3.3.4.

2.1.2 Výchozí brána

Uživatel může zásilku doručit osobně, pokud adresát bydlí poblíž, nebo ji může odnést na nejbližší poštovní úřad. Na jeho adresu se nejspíše zeptá někoho v okolí. V IP sítích je tato

adresa nazývana výchozí bránou. Pokud pro zásilku není známa jiná preferovaná cesta, použije se se výchozí brána. Opět může adresu zadat administrátor, nebo jí uživatel obdrží z DHCP.

2.1.3 Hiearchie

Poštovním společnostem se osvědčilo využívat určitou hiearchizaci adres, takže pokud poštovní úředník v Moskvě narazí na balík určený do České republiky, nebude již dále zkoumat databázi ruských měst, zda v nich není obsažena Ostrava, ale pošle balík rovnou na centrální příjem zahraničních zásilek do České republiky. Zde je již podle PSČ balík odeslán do Ostravy, poté na místní pobočku dle adresy a poté doručen na cílovou adresu. V IP sítích je to podobné. Adresy jsou hiearchicky rozděleny podle *prefixu* - čím je číslo vyšší, tím konkrétnější je adresa. Poštovní uzel, který poštu přeposílá na ostatní uzly je reprezentován zařízením jménem *router*.

2.1.4 Databáze adres

Nyní již má uživatel jedinečnou adresu a chtěl by začít korespondenci s nějakým dalším uživatelem. Může mít k dispozici svůj vlastní seznam adres, ze kterého adresu vyhledá. Problém nastává, pokud se uživatel přestěhoval, nebo není odesilateli adresa příjemce známa. Místo toho může použít nějaký centrálně dostupný seznam, který je pokud možno co nejvíce aktuální. V IP sítích má uživatel jednoznačné, dobře srozumitelné doménové jméno[1], které je přeloženo na adresu pomocí DNS serveru. Systém DNS je podrobně popsán v kapitole 6.

2.1.5 Bezpečnost

Než poštovní úředník předá adresátovi balík, většinou po něm požaduje nějaké ověření identity. Také se předpokládá dodržení listovního tajemství při putování zprávy. Ve světě počítačových sítí je toto ověřování celkem rozsáhlé a lze na něm postavit i další návazné služby spojené s bezpečností. Zabezpečení počítačových sítí jsou věnovány rozsáhlé publikace a v příslušných odděleních pracuje velké množství počítačových odborníků. Základnímu zabezpečení sítě pomocí protokolu RADIUS je věnována kapitola 7.

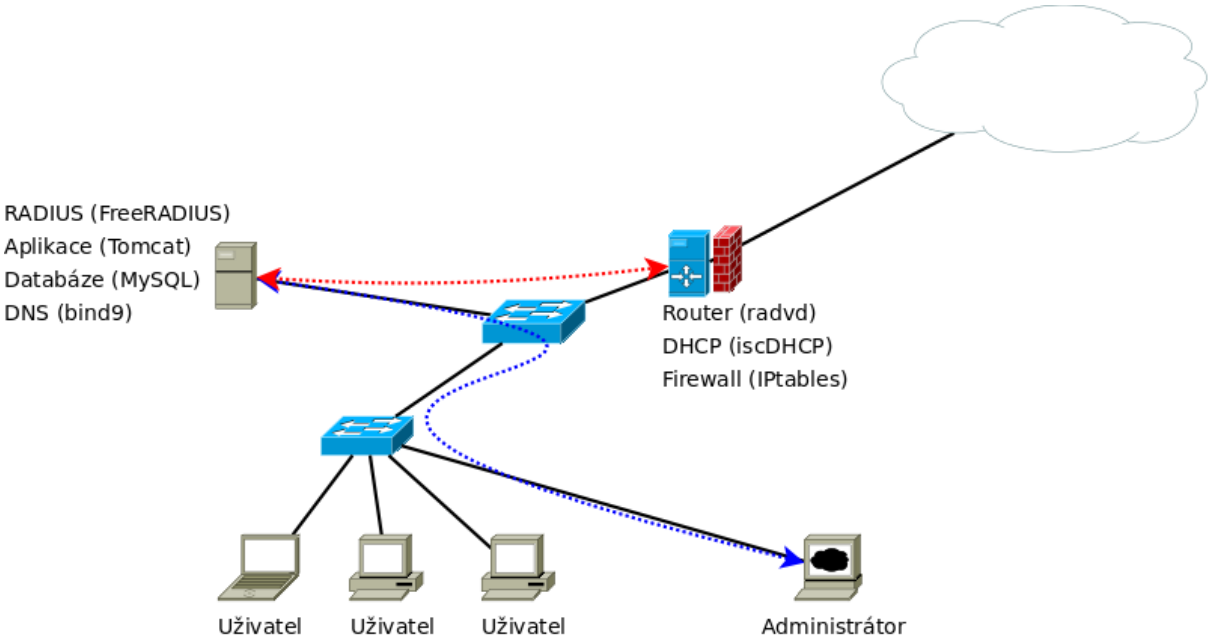
2.1.6 Další služby

Co když se uživatel stěhuje, nebo cestuje a potřebuje aby jej pošta vždy dostihla? Na své nejbližší pobočce zažádá o přeposílání na určitou adresu. V IPv6 sítích je tato funkce nazývaná *roaming* a její popis je mimo rozsah této práce. Bližší popis je možné nalézt v knize [2]. Do IPv4 sítí je tato možnost zpětně zavedena, není ovšem příliš využívaná.

Příklad topologie

Topologie jednoduché sítě s výše popisovanými službami by mohla být například ta na obrázku 1 na str. 6: Na obrázku administrátor pohodlně konfiguruje aplikaci ze svého pracovního PC, zatímco aplikace samotná se již postará o rozšíření konfigurace na potřebné zařízení.

[1] Až na vyjímky typu: *prvni-nejdelsi-domena-v-ceske-republice-ktera-ma-prave-63-znaku.cz*



Obrázek 1: *Příklad topologie*

2.2 Potřeby administrátora sítě

Každá síť potřebuje někoho kdo jí bude udržovat v chodu. Topologie sítě zpravidla není definitivní, ale mění se a vyvíjí podle potřeb uživatelů nebo provozovatelů. Může se jednat o rozšiřování sítě o nové uživatele, výměnu zařízení za kvalitnější a výkonnější anebo o nové nařízení z vedení společnosti ohledně vyššího zabezpečení, ... Příčin může být nespočet. Proto by měla existovat zodpovědná osoba (nebo skupina osob), která bude mít všechny tyto činnosti vykonávat - administrátor sítě. Ten může prvky sítě (uživatelské stanice i hardware určený k přenosu dat) konfigurovat několika způsoby. Pokusím se nyní shrnout jednotlivé přístupy k řešení, jejich výhody a nevýhody a použitelnost.

2.2.1 Ruční konfigurace

Administrátor může obejít všechny stanice a nastavit na nich všechny síťové parametry ručně. Je to velice zdlouhavá práce, která vyžaduje naplánování topologie sítě dopředu. Tato metoda je dnes použitelná pouze v opravdu malých sítích, nebo pro speciální účely. Výhodou je nezávislost na centrálním prvku konfiguračním - takováto síť bude fungovat, i bez DHCP serveru. Proto je tento přístup často použit na kritických síťových prvcích (routery, servery, ...). Nevýhodou je samozřejmě špatná flexibilita v případě konfiguračních změn.

2.2.2 Centrální konfigurace

Administrátor spravuje parametry sítě na jednom místě. Parametry jsou poté automaticky přiděleny na základě pravidel uživatelům. Velká výhoda je přehlednost stavu sítě a flexibilita. Např. pokud dojde ke změně výchozí brány pro segment sítě, nemusí administrátor měnit nastavení na všech stanicích, ale pouze na jednom místě v konfiguraci. Administrátorovi se tak také dostává do rukou silný nástroj, pomocí kterého může kontrolovat a dynamicky

měnit přístupy k různým síťovým zdrojům. Nevýhodou je samozřejmě závislost uživatelů na funkčnosti serveru poskytujícího tyto služby.

2.2.3 Distribuovaná centrální konfigurace

Tento přístup spočívá v rozložení služeb nutných pro funkčnost sítě na více zařízení, kdy na každém běží jedna (nebo i více) služeb nezávisle na ostatních - například DHCP server neběží na stejném zařízení jako DNS server. Zůstává otázkou, jak nakonfigurovat tyto distribuované servery. Jedna možnost je úprava konfiguračních souborů na všech serverech při každé změně. Poté se opět dostáváme ke stejnému problému s ruční konfigurací mnoha zařízení, pouze tentokrát se nejdná o stanice uživatelů, ale o síťové prvky. Proto jsem došel k návrhu aplikace, která má za úkol z jednoho místa pohodlně spravovat seznamy uživatelů, jejich zařízení a přidělených síťových parametrů. Aplikace tedy umí nakonfigurovat ostatní servery poskytující síťové služby. Současně ale při případném dočasném výpadku aplikace, bude síť fungovat dále - zbavíme se tak částečně úzkého místa ve funkčnosti sítě.

3 IP

Každé zařízení na sítí má nějaký jednoznačný identifikátor. Většina dnešních sítí, včetně všech počítačů na Internetu, používá TCP/IP protokol jako standart pro komunikaci po síti. V protokolu TCP/IP je tímto jednoznačným identifikátorem právě IP adresa.

3.1 Co je to IP adresa?

IP adresa je číslo, které jednoznačně identifikuje síťové rozhraní v počítačové síti používající IP (internetový protokol). Designéři internetového protokolu definovali IP adresu jako 32-bitové číslo a tento systém, známý jako IPv4 je běžně užíván dodnes. V důsledku enormního rozšíření Internetu a předpovězeného vyčerpání dostupných adres, byl navržen v roce 1995 nový adresní systém IPv6. Přestože se o příchodu systému IPv6 mluví již hodně dlohou dobu, není jeho rozšíření v menších sítích zatím příliš běžné. Důvodem může být, že administrátoři berou konfiguraci IPv6 jako *něco navíc*, co není k provozu sítě nutné a co pouze *přináší komplikace*. Systém, který jsem navrhl, by měl tyto konzervativce přesvědčit svou jednoduchostí a hlavně jednotností rozrhaní, kdy konfigurace parametrů IPv4 a IPv6 je hezky pohromadě na jednom místě.

3.2 IPv4

IPv4 adresní pole sestává z 32 bitů. Běžně se ovšem rozdělí do čtyř bloků po osmi bitech = oktetech. Každý ze čtyř oktetů převedeme do dekadické soustavy a oddělíme tečkou (*dot*). Proto se této notaci také říká *dotted decimal*. Příkladem takové adresy může být `192.0.2.73` jako na obrázku 2 na str. 8. Základní myšlenkou bylo, že každé zařízení připojené k internetu bude mít svojí jedinečnou IPv4 adresu.

V průběhu času zjišťovaly společnosti připojené k internetu, že je třeba jejich vnitřní sítě zabezpečovat. Jedna z oblíbených možností byla rozdělit adresní rozsah na veřejnou a neveřejnou část. Tento trend hezky korespondoval se snahou IETF o šetření IP adresami. Proto IETF přišla se sadou revizí, kteté vyústily v RFC 1918, která určila následující rozsahy jako privátní. [1]

Obrázek 2: *Struktura adresy IPv4, zdroj [1]*

Privátní rozsahy adres:

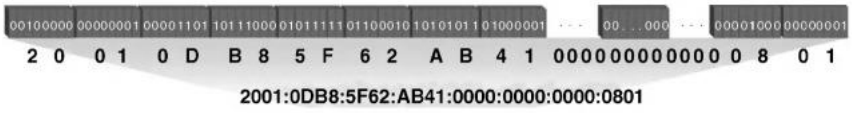
```
10.0.0.0 - 10.255.255.255 (10.0.0.0/8)
172.16.0.0 - 172.31.255.255 (172.16.0.0/12)
192.168.0.0 - 192.168.255.255 (192.168.0.0/16)
```

Význam privátních je, že tyto adresy nejsou směrovatelné v Internetu[2]. Proto nejsou tyto rozsahy adresovatelné z veřejné sítě → komunikaci je možné zahájit pouze směrem z privátní sítě do veřejné. Toto lze považovat za výhodu i nevýhodou současně. Z hlediska bezpečnosti je to určitě přínos, ale z pohledu uživatelů je to často chápáno jako omezení. Služba, zajišťující přemapování privátních adres na veřejné se nazývá NAT (Network Address Translation).

3.3 IPv6

Vzpomeňme si na přepis IPv4 adres do formátu čtyř čísel oddělených tečkami. Pokud Vám tento formát přišel obtížně zapamatovatelný, IPv6 Váš život ještě ztíží. IPv6 adresy jsou připisovány do formátu osmi 16-ti bitových čísel, která jsou v hexadecimálním tvaru a oddělená dvojtečkami. Každý hexadecimální znak reprezentuje čtyři bity mapované na znaky 0-F. Postup je znázorněn na obr. 3 na str. 8. Následuje seznam možných mapování:

0 = 0000	4 = 0100	8 = 1000	C = 1100
1 = 0001	5 = 0101	9 = 1001	D = 1101
2 = 0010	6 = 0110	A = 1010	E = 1110
3 = 0011	7 = 0111	B = 1011	F = 1111



Obrázek 3: *Struktura adresy IPv6, zdroj [1]*

3.3.1 Druhy IPv6 adres

V IPv6 existují tři druhy adres s odlišným chováním

- **Individuální (unicast)** jsou staré známe krotké adresy. Každá z nich identifikuje jedno síťové rozhraní a data mají být dopravena právě jemu.

[2]Lze je sice zabalit do tunelu, ovšem oba konce tunelu musí mít opět veřejné IP adresy

- **Skupinové (multicast)** slouží pro adresování skupin počítačů či jiných zařízení. Pokud někdo odešle data na tuto adresu, musí být doručena všem členům skupiny.
- **Výběrové (anycast)** představují novinku a nejzajímavější přírůstek v IPv6. Také výběrové adresy označují skupinu, data se však doručí jen jedinému jejímu členovi – tomu, který je nejblíže. Tyto adresy mohou být velice užitečné např. při hledání nejbližších dostupných DNS serverů.

Porovnání s IPv4 ukazuje, že zmizely oznamovací (broadcast) adresy. Nejsou potřeba, protože jejich funkce přebírají adresy skupinové. Jsou definovány speciální skupiny, např. pro všechny uzly na dané lince, které nahrazují původní oznamování.[2]

3.3.2 Typy adres

Obrovský adresní prostor, který má IPv6 k dispozici, byl rozdělen do několika skupin - typů adres. Každý typ sdružuje adresy se společnou charakteristikou.[2]

prefix	význam
::/128	nedefinovaná adresa
::1/128	smyčka (loopback)
fc00::/7	unikátní individuální lokální
fe80::/10	individuální lokální linkové
ff00::/8	skupinové adresy
ostatní	individuální globální

3.3.3 Stavové (stateful) adresy

Stavová odrůda není nic nového pod sluncem. Jedná se o konfiguraci prostřednictvím DHCPv6 - počítač rozešle dotaz a DHCP server mu v odpovědi sdělí vše, co by o zdejší síti měl vědět. Takhle to dnes funguje zcela běžně v řadě IPv4 sítí. V síti IPv6 však nastala zásadní změna, konfigurace klienta pomocí DHCPv6 je pouze doplňující mechanismus. Primární informace o síti se klient dozví z ohlášení směrovačů (Router Advertisment - RA). Takže v DHCPv6 nelze klientovi nakonfigurovat prefix podsítě (subnet mask), ani implicitní směrovač (default router). Přidělení IPv6 adresy protokolem DHCPv6 je volitelné, směrovač v síti určuje pomocí inzerovaných příznaků, zda má klient DHCPv6 použít pro konfiguraci adresy (příznak M=1 v RA) nebo pouze pro konfiguraci parametrů sítě (bezstavové DHCPv6, příznak M=0 a O=1 v RA). Typické parametry, které musíme klientovi předat v DHCPv6 jsou IPv6 adresy DNS serverů (rekurzivních resolverů) a implicitní doménová přípona (domain search suffix).[21]

3.3.4 Bezestavové (stateless) adresy

Kreativní novinkou je konfigurace bezstavová (stateless autoconfiguration), která pro určení IP adresy nevyžaduje žádné servery. Základní myšlenka je celkem prostá: každý směrovač v určitých intervalech rozesílá do sítí, k nimž je připojen, takzvané ohlášení směrovače. V něm jsou obsaženy základní informace - především prefixy adres dané sítě a zda on sám může sloužit pro předávání paketů ven (jako implicitní směrovač, default gateway).

Z ohlášení směrovačů (o které může při startu aktivně požádat pomocí výzvy směrovači se počítač dozví, jaké adresy používá zdejší síť. K nim si doplní identifikátor rozhraní (typicky 64 bitů), který si jednoznačně vygeneruje ze své ethernetové adresy. Tak získá platné IPv6

adresy pro své rozhraní. Jejich jednoznačnost ověří pomocí detekce duplicit - pomocí výzvy sousedovi se dotáže, zda vytvořenou adresu již někdo nepoužívá. Dostane-li kladnou odpověď, nesmí adresu svému rozhraní nastavit a automatická konfigurace skončí neúspěšně.

V rámci bezstavové konfigurace si stroj také vytvoří základ směrovací tabulky - seznam implicitních směrovačů, kterým bude předávat pakety směřující mimo síť. Pokud je jich víc, zpočátku je prostě střídá a směrovací tabulku si postupně vylepšuje na základě jejich upozornění (přesměrování), pokud paket k určitému cíli poslal nevhodným směrovačem.

Zní to lákavě, přesto však má bezstavová konfigurace dost velkou pihu krásy. Vůbec totiž neřeší otázku DNS[3]. Počítač se z ní nedozví IPv6 adresy zdejších DNS serverů a jak již bylo řečeno, bez DNS se ve světě IPv6 žije velmi těžko. Existuje několik návrhů, například definovat pevnou standardní adresu pro lokální DNS servery, či doplnit informaci o nich do ohlášení směrovače. Experimenty jsou prováděny také s použitím tzv. anycast adres. Zatím ale jedinou spolehlivou cestou je použít bezestavové DHCPv6 doplňující klientovi další informace, jako právě adresy DNS serverů. [21]

4 Navržená aplikace

Aplikace, kterou jsem navrhl a implementoval má za úkol sjednotit správu více systémů (které mohou být distribuované na různých místech sítě) do jednoho centrálního konfiguračního rozhraní. Její primární určení bylo pro snažší implementaci IPv6 do stávajících (nebo nově vznikajících) IPv4 sítí. Vzal jsem tedy obě verze IP protokolu v úvahu už od začátku. Administrátor tedy může parametry pro obě verze protokolu zadávat pohodlně na jednom místě (např. přiřadit DNS servery verze 4 i verze 6) a systém sám rozhodne, do kterého konfiguračního souboru bude hodnota zapsána.

4.1 Požadavky na aplikaci

Před tím, než jsem začal s návrhem aplikace, bylo nutné určit požadavky které musí splňovat. K jejich splnění jsem se snažil co nejvíce přiblížit a myslím, že výsledek je rozhodně uspokojující. Požadavky tedy byly následující:

- **Spolehlivost** - Aplikace musí být schopna pracovat bez poruchy po dlouhé období. Její chování by mělo být předvídatelné a měla by být schopna odchytit případné špatné konfigurační parametry ještě před jejich nasazením do produkčního prostředí. S tímto požadavkem jsem se snažil vypořádat co nejlépe, ale předpokládám, že při nasazení do praxe a každodenním používání se projeví drobné nedostatky, které bude ještě třeba doladit.

- **Rozšiřitelnost/Univerzálnost** - Aplikace by měla být navržena co nejvíce modulárně, aby bylo možné její rozšíření o další funkcionalitu. V dnešním světě překotného vývoje informačních technologií je velice těžké odhadnout, jaké nové požadavky vzniknou například za pět let. Proto jsem třídy v aplikaci navrhnul s určitou mírou abstrakce, kdy například třídy pro konfiguraci sobuborů dědí z jejich rodičovské třídy *Config* a jsou povinni implementovat některé funkce. I když tyto postupy přinášejí při implementaci spoustu práce navíc, myslím, že se pro budoucí rozšiřitelnost aplikace jsou nezbytné. Také jsem počítal

[3]Tento problém je řešen v RFC 6106.

s tím, že syntaxe některých konfigurací se může s budoucími verzemi serverových software měnit. Proto jsem některé parametry umístil do dynamických polí, kdy je možné definovat nejenom hodnoty proměnných, ale dokonce i proměnné samotné. Příklad může být vkládání jmenných serverů do konfigurace DHCP. Zde stačí konfigurační hodnotu *default-lease-time* zadat s jejím parametrem *600* a aplikace již sama poskládá dle platné syntaxe konfigurační soubor.

- **Přehlednost** - Snažil jsem se rozhraní i program samotný napsat smysluplně a přehledně, kdy většina prvků má vlastní určené okno pro nastavení parametrů. Pokud je v systému málo záznamů, může se zdát, že by bylo jednodušší vše „vecpat“ do jednoho místa. Ze zkušenosti s podobnými systémy ovšem vím, že při nárůstu počtu záznamů se systém stává nepřehledným a i načítání může trvat velmi dlouho.

- **Rozsah a omezení** - Pro aplikaci předpokládám nasazení v menších až středně rozsáhlých sítích, jmenovitě do 300 PC. Systém se nemůže rovnat velkým komerčním systémům, vyvíjených konkrétně pro tento účel. Na druhou stranu jsem přesvědčen, že v sítích, pro které je určen odvede díky své jednoduchosti na obsluhu a přívětivosti svou práci více než dobře a pomůže k rozšíření IPv6 i do míst, kde zatím nebylo nasazení tohoto protokolu plánováno.

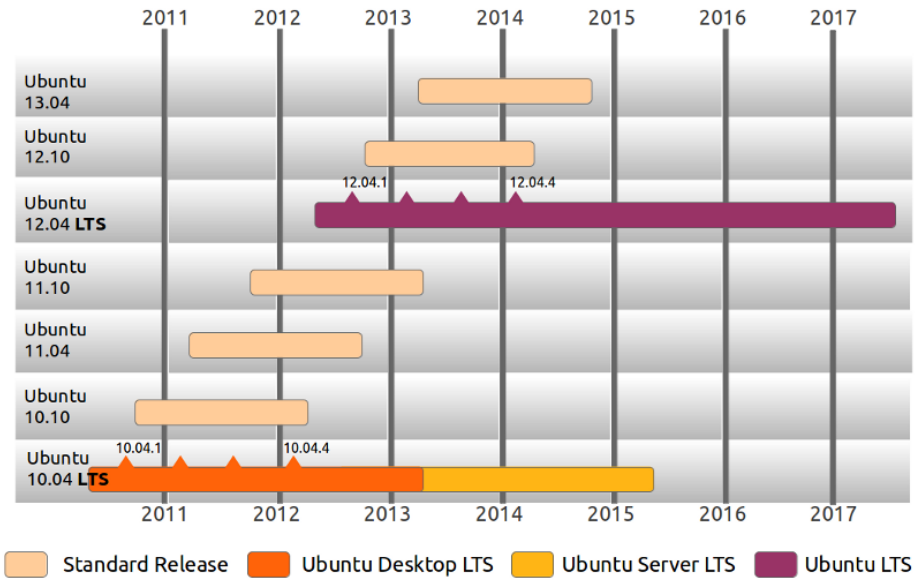
4.2 Operační systém

Linux je často chápán jako alternativní filosofie přístupu k výpočetní technice, která se točí okolo sdílení nejenom software, ale také znalostí. Tím, že začneme používat Linux se stáváme součástí mohutné globální komunity lidí, kteří se nechali zachytit fenoménem, který mění svět.

Ubuntu[22] je přirozeným vývojem těchto cílů. Jedná se o projekt založený podnikatelem Mark Shuttleworthem s úmyslem přinést volně dostupný a vysoce kvalitní operační systém do dosavadního převážně komerčního světa počítačů. K dnešnímu dni Shuttleworth investoval ze svých vlastních zdrojů 10 milionů dolarů aby zajistil provozuschopnost projektu. V roce 2011 se projekt již téměř dostal k hranici soběstačnosti díky skutečnosti, že se Ubuntu stál jedním ze tří hlavních operačních systémů instalovaných na uživatelská a serverová PC. [8]

Ubuntu je sestaveno okolo jedné z nejvíce „zaběhnutých“ verzí Linuxu: OS Debian. Projekt Debian odstartoval v roce 1993 (brzy poté co byla vypuštěna první verze Linux software) a stal se jedním z průkopnických variant Linuxu v mnoha ohledech. Ubuntu a Debian sdílejí většinu základních myšlenek, ovšem Ubuntu je více zaměřeno na desktopové protředí. Je tomu poměrně nedávno, kdy světlo světa spatřila verze dedikovaná pro servery. Tuto verzi jsem se rozhodl použít pro implementaci software. Není ovšem vyloučena možnost upravit software pro použití i na jiných distribucích.

Verzi distribuce Ubuntu jsem zvolil 10.04.3 LTS. Přestože jsou již k dispozici novější verze, tato má nespornou výhodu právě v příponě LTS. Ta totiž znamená Long Term Support (dlouhodobá podpora). Zatímco standartní distribuce vydávané každých 6 mesíců mají zaručenou podporu na 18 měsíců, distribuce označené LTS mají zaručeno, že se pro ně budou vyvíjet nové záplaty a rozšíření až do předem určeného roku - v případě 10.04 do dubna roku 2015. V době obhajoby této práce již bude vypuštěna nová LTS verze 12.04, bohužel dosud je pouze ve stádiu beta verze, kterou nepovažuji pro důkladné testování dostatečnou. Shrnutí verzí je na obr 4 na str. 12



Obrázek 4: *Ubuntu release cycle*

5 DHCP

5.1 Popis služby

DHCP (Dynamic Host Configuration Protocol) je protokol určený ke konfiguraci síťových parametrů uživatelů v IP sítích. Pokud chtějí počítače, které jsou do IP sítě připojené, komunikovat s ostatními, musí být nejdříve vhodně nakonfigurovány. Základní nutností pro komunikaci je IP adresa a výchozí brána. DHCP ulehčuje administrátorům život tím, že není třeba tyto hodnoty nastavovat ručně, ale jsou přidělovány z centrálního servereu. Další výhodou je, že databáze je na jednom místě, čímž se zamezí duplicitě IP adres v síti.

Navíc může DHCP také poskytnout další užitečné údaje, jako jsou adresy DNS serverů, network boot serverů a dalších.

DHCP je používáno v IPv4 i IPv6 sítích a přestože u obou verzí slouží ke stejnému účelu, detaily protokolu jsou natolik odlišné, že jsou někdy popisovány dva zcela různé protokoly.

DHCP bylo poprvé verifikováno pomocí RFC 1531 v roce 1993, jako rozšíření Bootstrap protokolu. Důvodem bylo omezení Bootstrap, kdy klientská rozrhaní musela být nejprve předkonfigurována, než mohla obdržet informace. Bootstrap také neposkytoval žádný mechanismus pro označení adres, které již nejsou využívány.

5.2 Stavová/Bezestavová konfigurace

Základní rozhodování v IPv6 je, zda použít *stavové* (*stateful*) 3.3.3 nebo *bezestavové* 3.3.4 (*stateless*) adresy. Bezestavová konfigurace je příjemná svou jednoduchostí, většinou nemusí klient nic nastavovat a IPv6 adresu vygeneruje sám . Problém však vzniká se získáním adres

DNS severů. Zde může buď v síti figurovat DHCP server, který tyto informaci klientu předá, nebo si je může klient nastavit ručně.

V případě použití stavové konfigurace je v síti nutná přítomnost DHCP serveru, který klientovi přidělí IPv6 adresu i všechny ostatní informace potřebné pro funkční připojení. V tomto případě vzniká pochopitelně možnost klienty lépe spravovat a kontrolovat. Je možné i kombinovat oba přístupy, kdy se „známým“ klientům přiřadí pevné adresy a „neznámým“ se ponechá automatická konfigurace.

Rozhodl jsem se ve svém systému zaměřit na stavové přidělování adres, protože většina administrátorů si přeje mít přehled nad tím, kdo se do sítě připojuje.

5.3 DHCPv4 server

Nejrozšířenějším DHCPv4 serverem pro OS Linux je jednoznačně implementace iscDHCP. Osobně mám s tímto serverem dobré zkušenosti. Pokud je správně nakonfigurován, nikdy se mi zatím nestalo, že by „zamrzl“ nebo dokonce neočekávaně skončil. Proto jsem se pro IPv4 rozhodl pro tento server.

5.4 DHCPv6 server

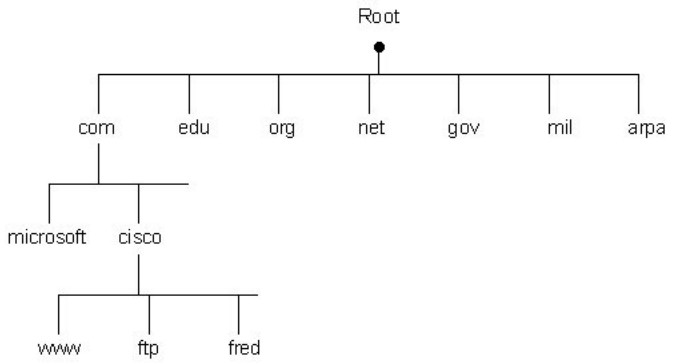
Na použitém operačním systému4.2 stojí v nabídce balíčku pro DHCPv6 za zmínku *wide-dhcpv6-server*[7], popř. *dibbler-server*[12]. Bohužel ani jeden z těchto nesplnil moje očekávání ohledně možností konfigurace, podpory a stability. Wide DHCP server má problémy s podporu klientů na jiných operačních systémech než Linux, Dibbler je zatím spíše v rané fázi vývoje a má velké množství chyb. Moje pozornost se tedy zaměřila na *isc-dhcp-server*[6]. Ten je pro Ubuntu 10.04 dostupný ve verzi 3 pod názvem *dhcp3-server*. Ta ovšem nepodporuje IPv6, ale pouze IPv4. V dalších vydáních Ubuntu (od 11.04) je dostupná novější verze *4*, která práci s IPv6 zvládá celkem obstojně, přestože její vývoj ještě není dokončen. Abych zaručil kompatibilitu skriptů s budoucími verzemi OS, rozhodl jsem se stáhnout zdrojové kódy poslední dostupné verze přímo pro ISC a sám tuto novější verzi zkompilovat. Obrovskou výhodou *isc-dhcp* je totiž jeho stávající rozšířenost pro IPv4. Je tedy velká pravděpodobnost, že administrátor se již s konfigurací setkal. Navíc je snadné provozovat na jednom serveru IPv4 i IPv6 DHCP služby, stačí spustit dvě různé instance s přepínačem -4/-6 a odpovídajícím konfiguračním souborem.

6 DNS

6.1 Popis služby

Pro komunikaci v naprosté většině dnešního internetu, je třeba znát **IP adresu**. Jsou to jednoznačné identifikátory, určující cíl, kam budou informace putovat. **Doménová jména** jsou srozumitelné a snadno zapamatovatelné ukazatele na IP adresy. Není reálné, aby si uživatelé pamatovali IP adresy pro všechny stránky které navštěvují (a také pro e-mailové servery a spoustu dalších služeb). Navíc by s tímto přístupem byly další problémy, protože IP adresy příslušné některým službám se často mění. **DNS (Domain Name System)** je služba, pomocí níž jsou doménová jména dohledána a přeložena na IP adresy. Se znalostí IP adresy se již aplikace může pustit do rozesílání paketů určených cíli.

Protože udržování centrální databáze všech mapování doménových jmen na IP adresy není praktické (a nejspíše ani realizovatelné), jsou seznamy doménových jmen a příslušných adres distribuovány po internetu hiearchií autorit. Nejvyšší autorita je „.“, která se dále dělí na tzv. TLD (top-level domain). Viz obrázek 5 na str. 14



Obrázek 5: *Hiearchie DNS, zdroj [19]*

Aby mohl uživatel používat DNS servery, musí znát IP adresu alespoň jednoho z nich (většinou se konfigurují minimálně dva). Tento server se nazývá lokální a bývá umístěn u poskytovatele připojení a funguje jako rekurzivní6.4. Adresu lokálního serveru počítač typicky obdrží prostřednictvím DHCP. Pokud počítač hledá určitou informaci v DNS (např. IP adresu k danému jménu), obrátí se s dotazem na tento lokální server. Každý DNS server má ve své konfiguraci uvedeny IP adresy kořenových serverů (autoritativních serverů pro kořenovou doménu). Obrátí se tedy s dotazem na některý z nich.

Kořenové servery mají autoritativní informace o kořenové doméně. Konkrétně znají všechny existující domény nejvyšší úrovně a jejich autoritativní servery. Dotaz je tedy následně směrován na některý z autoritativních serverů domény nejvyšší úrovně, v níž se nachází cílové jméno. Ten je opět schopen poskytnout informace o své doméně a posunout řešení o jedno patro dolů v doménovém stromě. Tímto způsobem řešení postupuje po jednotlivých patrech doménové hierarchie směrem k cíli, až se dostane k serveru autoritativnímu pro hledané jméno, který pošle definitivní odpověď. [26]

6.2 DNS záznamy

6.2.1 Zónové soubory

Obsahují definici domény. Skládá se z více záznamů, jejichž názvy a obsah jsou přesně definovány (*RFC 1035*). Formát se může lišit dle použitého serveru. V našem případě si ukážeme konfiguraci pro software jménem BIND [13]

Jako příklad vybereme existující záznam, který je příslušný doméně „janmichalek.cz.“.

DNS záznam pro doménu janmichalek.cz

```
janmichalek.cz. 3600 IN SOA ns.banan.cz. domeny.banan.cz. 1330979267
   14400 3600 1209600 3600
janmichalek.cz. 3600 IN NS ns.banan.cz.
janmichalek.cz. 3600 IN NS ns2.banan.cz.
janmichalek.cz. 3600 IN NS ns3.banan.it.
janmichalek.cz. 3600 IN MX 5 p.banan.cz.
janmichalek.cz. 3600 IN MX 10 backup.p.banan.cz.
*.janmichalek.cz. 3600 IN A 77.93.211.241
@ 3600 IN A 77.93.211.241
s.janmichalek.cz. 3600 IN A 84.42.223.253
*.janmichalek.cz. 3600 IN AAAA 2a01:430:1a::241
@ 3600 IN AAAA 2a01:430:1a::241
pop3.janmichalek.cz. 3600 IN CNAME p.banan.cz.
imap.janmichalek.cz. 3600 IN CNAME p.banan.cz.
smtp.janmichalek.cz. 3600 IN CNAME p.banan.cz.
s2.janmichalek.cz. 3600 IN CNAME houbicka.dyndns.org.
janmichalek.cz. 3600 IN TXT "v=spf1 a mx a:relay.p.banan.cz -all"
```

6.2.2 Typy DNS záznamů

Formát záznamů:

Doménové jméno	životnost	třída	typ	data
`janmichalek.cz.`	3600	IN	NS	`ns.banan.cz.`

- Doménové jméno - Pokud není ukončeno tečkou, doplní se aktuální doména.
  - `*.janmichalek.cz.` - všechny záznamy daného typu, které nejsou jinde definovány
  - `@` - doplní název domény (Origin)
- Životnost - Pokud se tento záznam nachází v cache paměti nějakého serveru, je po uplynutí této doby považován za neplatný
- Třída - Zpravidla internet (IN)
- Typ
  - **SOA** - (State Of Authority)
    Obsahuje jméno primárního serveru, adresu elektronické pošty jejího správce a dále různé časovače .
    `janmichalek.cz. 3600 IN SOA domeny.banan.cz. 1330979267 14400 3600 1209600 3600`
  - **NS** - (Name Server)
    Jméno autoritativního DNS serveru
    `janmichalek.cz. 3600 IN NS ns.banan.cz.`
  - **MX** - (Mail Exchange)
    Adresa a priorita e-mailového serveru. Nižší číslo udává vyšší prioritu.
    `janmichalek.cz. 3600 IN MX 5 p.banan.cz.`

- **A** - IPv4 adresa příslušná k doménovému jménu
  `s.janmichalek.cz. 3600 IN A 84.42.223.253`
- **AAAA** - IPv6 adresa příslušná k doménovému jménu
  `*.janmichalek.cz. 3600 IN AAAA 2a1:430:1a::241`
- **CNAME** - (Canonical Name) alias pro již zavedené jméno
  `pc.janmichalek.cz. 3600 IN CNAME www.janmichalek.cz.`
- **TXT** - (Human Readable Text)
  Dříve sloužil pro „poznámky“, nyní jej používají i další služby (např. SPF) [11]
  `janmichalek.cz. 3600 IN TXT "v=spf1 a mx a:relay.p.banan.cz -all"`
- **PTR** - (Pointer)
  Speciální typ záznamu pro reverzní zóny. Na levé straně obsahuje adresu a na pravé jméno počítače.

6.3 Typy DNS serverů

6.4 Rekurzivní DNS servery

Pokud takovýto server obdrží dotaz, pokusí se sám zjistit potřebné informace a tazateli předá až konečný výsledek. Často také výsledek uloží do cache6.7, aby nemusel opětovně zjišťovat stejné informace.

6.5 Nerekurzivní DNS servery

Aktivně nezjišťují potřebné informace, pouze předávají dotaz dalším DNS serverům.

6.6 Reverzní dotazy

Nejběžnějším úkolem DNS je poskytnout informace (nejčastěji IP adresu) pro zadané doménové jméno. Dovede ale i opak – sdělit jméno, pod kterým je daná IP adresa zaregistrována. Při vkládání dat pro zpětné dotazy bylo ale třeba vyřešit problém s opačným uspořádáním IP adresy a doménového jména. Zatímco IP adresa má na začátku obecné informace (adresu sítě), které se směrem doprava zpřesňují až k adrese počítače, doménové jméno má pořadí přesně opačné. Instituce připojená k Internetu typicky má přidělen začátek svých IP adres a konec svých doménových jmen.

Tento nesoulad řeší DNS tak, že při reverzních dotazech obrací pořadí bajtů v adrese. K obrácené adrese pak připojí doménu in-addr.arpa a výsledné „jméno“ pak vyhledává standardním postupem. Hledá-li například jméno k IP adrese `77.93.211.241`, vytvoří dotaz na `241.211.93.77.in-addr.arpa.` Protože IP adresu `77.93.211.241` má ve správě ISP „Master Internet“, obsahuje odpověď na reverzní dotaz SOA záznam:

```
211.93.77.in-addr.arpa. 2182 IN SOA ns1.westmaster.com. zidek.master.cz. 2011121800
3600 3600 1209600 3600
```

6.7 Cache

Téměř každý DNS server funguje zároveň jako DNS cache. Při opakovaných dotazech pak nedochází k rekurzivnímu prohledávání stromu, ale odpověď je získána lokálně. V DNS záznamech je totiž uložena i informace jak dlouho lze záznam používat (TTL) a lze také zjistit, zda byl záznam změněn. Po vypršení platnosti je záznam z DNS cache odstraněn.

6.8 DNS servery dle autorit

- **Master**
  Dle starší notace Primary. Server, na který byla delegována správa zóny. Obsahuje konfigurační soubory pro danou zónu. Při obdržení dotazu na doménu, která je v jeho správě, odpoví autoritativní odpovědí. Zpravidla se chová jako nerekurzivní6.5, aby zamezil (D)DoS útokům. [5] Pokud dojde ve změně v jeho konfiguraci, může vygenerovat oznámení pro Slave servery.

- **Slave**
  Dle starší notace Secondary. Kopíruje data z Master serveru pomocí transferu zóny. Odpovídá autoritativně, není proto možné rozeznat, zda odpověď na DNS dotaz přišla od serveru typu Master nebo Slave.

- **Stub**
  V definici zóny obsahuje pouze informace, jaké servery jsou pro danou zónu autoritativní.

- **Forwarding**
  Chová se jako nerekurzivní6.5 a cache server6.7: Přepošle dotazy na jiný server a ukládá výsledky do cache.

6.9 Konfigurace DNS

Aplikace využívá DNS server Bind ve verzi `1:9.7.0.dfsg.P1-1ubuntu0.4` dostupnou z repozitářů. Abych oddělil statickou konfiguraci, která nebude měněna, od konfigurace generované aplikací, vytvořil jsem v konfiguračním adresáři`/etc/bind/` následující strukturu:

`system/named.conf.local`	Soubor obsahující definici zónových souborů
`system/zones/master/`	Adresář se zónovými soubory

Dále jsem přidal do globálního konfiguračního souboru `/etc/bind/named.conf` následující řádek, který zajistí, že přidané soubory budou použity.

Kód:

```
include "/etc/bind/system/named.conf.local";
```

Protože systém bude použit pro překlad veškerých doménových jmen v přidělené síti, nastavil jsem jej jako master6.8 a forwarding6.8 současně. Cílový server pro přeposílání dotazů na nenalezené záznamy vybere administrátor při počáteční konfiguraci aplikace Bind. Pro testovací účely jsem použil veřejně dostupný DNS resolver společnosti Google na IP `8.8.8.8`

Na tomto místě bych ještě rád zmínil možnost konfigurace, kterou nabízí framework MySQL Bind SDB Driver. Jedná se o přímé napojení aplikace Bind do MySQL serveru, kdy není při změně třeba reload nastavení. Změna se provede pouze v databázi a projeví se hned při příštím dotazu. Systém jsem nepoužil, protože jej nepovažuji za dostatečně robustní pro poskytování DNS služeb. Při výpadku databáze by byl celý systém ochromen, zatímco u mé aplikace by pouze nebylo možné se záznamy pracovat, zavedené záznamy však budou fungovat nadále. Navíc z vlastní zkušenosti vím, že Bind server s cca 13000 doménami uloženými na filesystému (včetně konfigurací DNSSEC záznamů) provede kompletní reload za necelou

minutu. Když tuto hodnotu srovnáme s dobou, po kterou jsou záznamy drženy v cache DNS serverů (rámcově hodiny), jeví se doba reloadu jako nepodstatná. Rozhodně nám tedy vychází lépe přístup, který jsem použil.

7 RADIUS

7.1 Popis služby

Remote Authentication Dial In User Service (RADIUS) je protokol pro konrolu přístupu, který ověřuje a autentizuje uživatele na základě challenge - response metody.[3] V případě ISO OSI modelu pracuje na čtvrté vrstvě a využívá UDP paketů. Poskytyje centralizovanou autentizaci, autorizaci a účtování (AAA)7.2 jako nástroj správy počítačů, které se připojují k síti a používají ji. RADIUS byl vydán společnosti Livingston Enterprises, Inc. v roce 1991. Z důvodu široké podpory a všudypřítomného provázání RADIUS protokolu, je často používán poskytovateli Internetového připojení a velkými společnostmi k řízení přistupu k Internetu nebo interním sítím. (Včetně bezdrátových, DSL, VPN apod.).

RADIUS je protokol pracující na modelu klient/server. Remote Access Server (RAS), Virtual Private Network Server (VPN), přepínač s autentizací na bázi portů a Network Access Server - To vše jsou zařízení, které umožňují přístup k jiným sítím a všechny mají zabudovány klientské komponenty pro komunikaci s RADIUS serverem. Implementace protokolu - RADIUS server bývá většinou spuštěn jako démon na pozadí UNIX nebo Microsoft serverů.

7.2 AAA

V oblasti počítačové bezpečnosti AAA znamená authentication, authorization and accounting protocol, tj. česky autentizační, autorizační a účtovací protokol.

7.2.1 Autentizace

Autentizace (Authentizaction) znamená potvrzení, že uživatel požadující služby je platným uživatelem poskytovaných síťových služeb. Autentizace je dosažena pomocí představení identity a jistého pověření nebo tajemství. Mezi různé typy tajemství patří například hesla, pověření na jedno použití, digitální certifikáty nebo telefonní čísla (ať už volající nebo volaná).

7.2.2 Autorizace

Autorizace (Authorization) znamená udělení specifického typu služby (včetně „žádné služby“) uživateli, na základě jeho autentizace, služeb, které požaduje a aktuálního stavu systému. Autorizace může být založena na omezeních, například omezení na určité hodiny v rámci dne, nebo omezení na fyzickou polohu, nebo omezení vícenásobného přihlášení jednoho uživatele. Autorizace určuje povahu služby, která je poskytnuta uživateli. Typy služeb jsou například: filtrování IP adres, přidělení adresy, přidělení cesty, QoS, řízení šířky pásma/řízení toku, tunelování do konkrétního koncového bodu, nebo šifrování.

7.2.3 Účtování

Účtování (Accounting) znamená sledování využívání síťových služeb uživateli. Tyto informace mohou být použity pro správu, plánování, účtování, nebo další účely. Účtování v reálném čase

je doručeno současně s využíváním zdrojů. Dávkové účtování ukládá informace o účtech dokud není později doručena. Běžně se sbírají informace o identitě uživatele, povaze dodaných služeb a časy počátků a konců dodaných služeb.

7.3 AAA v podání RADIUS

RADIUS používá AAA koncept k řízení přístupu k síti v následujcím procesu, který je rozdělen do dvou částí. Autentizace a autorizace jsou popsány v RFC 2865, zatímco účování je popsáno v RFC 2866.

7.3.1 Autentizace a autorizace

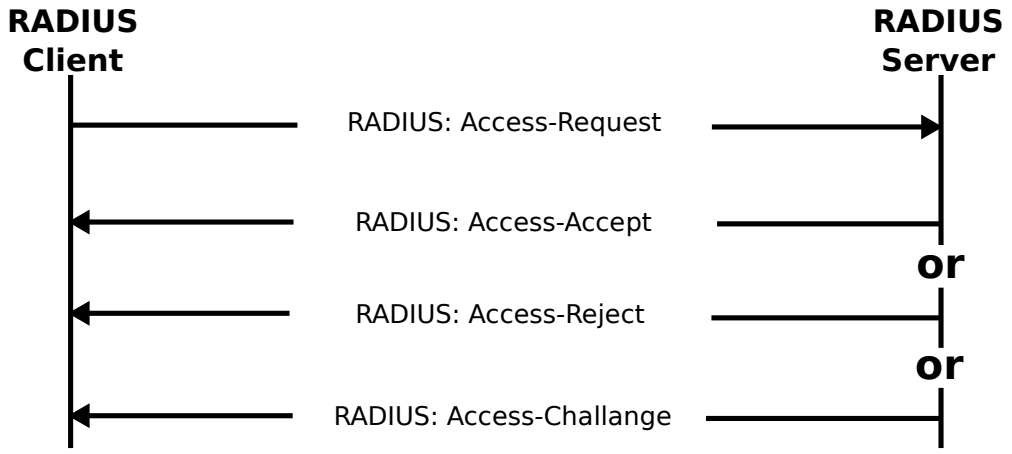
Uživatel zařízení zašle požadavek na RAS s tím, že chce získat přístup k určitému prostředku v sítí s použitím svých ověřovacích údajů. Tyto údaje jsou předány RAS zařízení pomocí protokolu na linkové vrstvě (např. PPP v případě mnoha DSL poskytovatelů).

Jako odpověď RAS zašle *RADIUS Access Request* zprávu na RADIUS server. Zpráva obsahuje požadavek na získání přístupu pomoci RADIUS protokolu.

Tento požadavek obsahuje ověřovací údaje, většinou ve formě uživatelského jména a hesla nebo bezpečnostního certifikátů, kterým se uživatel prokazuje. Navíc může RAS připojit i dodatečné údaje, jako je např. telefonní číslo nebo síťovou adresu.

RADIUS server poté ověří pravost údajů pomocí autentizačních schémat jako jsou PAP, CHAP nebo EAP. Pokud byly dodány dodatečné údaje spolu s jménem a heslem, jsou ověřeny také včetně privilegia k daným prostředkům.

V původním navrhu RADIUS zkontroloval uživatelské údaje v lokálním souboru (který sloužil jako jednoduchá databáze). Moderní RADIUS implementace umí pracovat i s externími zdroji, jako je SQL, Kerberos, LDAP nebo Active Directory. Pro účely aplikace jsem se rozhodl využít MySQL databázi pro ukládání uživatelských účtů i seznamu zařízení.



Obrázek 6: *Autentizační protokol. Zdroj [27]*

7.4 FreeRADIUS

FreeRADIUS je implementace RADIUS serveru, která je k dispozici jako open source pod licencí GPL. Jedná se o nejrozšířenější a nejpopulárnější implementaci.

7.4.1 Instalace pro IPv4

Nejdříve jsem použil instalace dostupné z repozitářů ubuntu. Verze dostupná v době psaní této práce je *freeradius_2.1.8+dfsg-1ubuntu1*. Následujícím postupem jsem zprovoznil AAA vůči MySQL databázi pro IPv4.

1. Instalace potřebných balíků

   Kód:

   ```
   # aptitude install freeradius freeradius-mysql
   ```

2. Vytvoření MySQL uživatele a schema databáze
   Administrátor zde vybere vlastní heslo do databáze místo jednoduchého *S3cret*.

   Kód:

   ```
   # mysql -u root -p
   CREATE DATABASE ‘radius‘ ;
   CREATE USER ’radius’@’localhost’ IDENTIFIED BY ’S3cret’;

   GRANT USAGE ON * . * TO ’radius’@’localhost’ IDENTIFIED BY ’
       S3cret’
   WITH MAX_QUERIES_PER_HOUR 0 MAX_CONNECTIONS_PER_HOUR 0
   MAX_UPDATES_PER_HOUR 0 MAX_USER_CONNECTIONS 0 ;

   GRANT ALL PRIVILEGES ON ‘radius‘ . * TO ’radius’@’localhost’ WITH
        GRANT OPTION;
   FLUSH PRIVILEGES;
   exit;

   # cd /etc/freeradius/sql/mysql
   # mysql -u radius -pS3cret radius < schema.sql
   # mysql -u radius -pS3cret radius < nas.sql
   ```

3. Připravíme si testovací uživatele pro testování spojení do databáze. Jeden se bude přihlašovat pomocí plaintext, druhý pomocí MD5 hashe. Zde je třeba dodržet správné operátory. V aplikaci jsou použity jenom následující dva:

   - „==“ Defaultní operátor při vložení do databáze. Pokud požadavek zaslaný uživatelem obsahuje daný atribut, je jeho hodnota ověřena.
   - „:=“ Stejná funkce jako předchozí. Navíc pokud je v konfiguraci atribut se stejným názvem, jeho hodnota je nahrazena. Pokud v dotazu od klienta není daný atribut přítomen, je do dotazu připojen.

   Detailní popis všech možných operátorů je možné nalézt na wiki aplikace FreeRADIUS [15]

Kód:

```
INSERT INTO radcheck (UserName, Attribute, Value)
VALUES ('sqlplain', 'Password', 'testpwd');
INSERT INTO radcheck (username,attribute,op,value)
VALUES ('sqlmd5', 'MD5-Password', ':=', MD5( 'Password' ));
```

4. Úprava konfiguračních souborů aplikace FreeRADIUS. Vypsány jsou pouze řádky, které byly změněny.

Kód:

```
# cd /etc/freeradius
# vim sql.conf
server = "localhost"
login = "radius"
password = "S3cret"
radius_db = "radius"
readclients = yes
```

Poslední řádek zapne načítání NAS klientů z databáze. Bohužel FreeRADIUS umí klienty načíst pouze při startu. Po úpravě tabulky je tedy třeba službu restartovat.

Kód:

```
# vim sites-enabled/default
authorize {
...
sql
...
}

authenticate {
...
sql
...
}

session {
...
sql
...
}
```

Pokud jsou uvedené porty *0*, aplikace FreeRADIUS by měla vybrat porty z */etc/services*. Bohužel se mi opakovaně stávalo, že FreeRADIUS odmítal nastartovat bez explicitně určených portů, proto jsou zde upraveny

Kód:

```
# vim radiusd.conf
listen {
  type = auth
  port = 1812
}
listen {
  type = acct
  port = 1813
}

\$INCLUDE sql.conf
```

5. Restart služby FreeRADIUS

Kód:

```
# /etc/init.d/freeradius restart
```

6. Vyzkoušíme lokálně uživatele, kterého jsme vytvořili. Adresa pro IPv4 localhost je standartně přidaná do souboru *clients.conf*, je tedy možné jí použít pro přístup. Standartní heslo je pro *localhost* nastaveno na *testing123*

Kód:

```
radtest sqlplain secr3t localhost 1812 testing123
radclient: socket: cannot initialize udpfromto: Function not
   implemented
```

Nyní se klient nemůže připojit, protože Ubuntu překládají *localhost* na IPv6 adresu ::1, na které neposlouchá server a současná verze klienta s ní neumí pracovat. (Přestože dokumentace tvrdí že ano, viz dále). Vyzkoušíme tedy přímo IPv4 adresu

Kód:

```
# radtest sqlplain secr3t 127.0.0.1 1812 testing123
Sending Access-Request of id 41 to 127.0.0.1 port 1812
        User-Name = "sqlplain"
        User-Password = "secr3t"
        NAS-IP-Address = 127.0.1.1
        NAS-Port = 1812
rad_recv: Access-Accept packet from host 127.0.0.1 port 1812, id
   =41, length=20

Sending Access-Request of id 183 to 127.0.0.1 port 1812
        User-Name = "sqlmd5"
        User-Password = "secr3t"
        NAS-IP-Address = 127.0.1.1
        NAS-Port = 1812
rad_recv: Access-Accept packet from host 127.0.0.1 port 1812, id
   =183, length=20
```

Zde vidíme, že spojení korektně funguje a autentizace proběhla.

7. Vložíme do tabulky *nas* záznam a vyzkoušíme i vzáleného IPv4 klienta. V případě testování jsem použil počítač s IP *10.0.0.101*. Heslo jsem nastavil stejné jako pro klienta *localhost*.

Kód:

```
INSERT INTO nas
  (id, nasname, shortname, type, ports, secret, community,
     description)
  VALUES
  (NULL, '10.0.0.101', NULL, 'other', NULL, 'testing123', NULL, '
     RADIUS Client');
```

A klienta vyzkoušíme:

Kód:

```
radtest sqlmd5 secr3t 10.0.0.1 1812 testing123
Sending Access-Request of id 125 to 10.0.0.1 port 1812
        User-Name = "sqlmd5"
        User-Password = "secr3t"
        NAS-IP-Address = 127.0.1.1
        NAS-Port = 1812
rad_recv: Access-Accept packet from host 10.0.0.1 port 1812, id
   =125, length=20
```

8. Vše je zde v pořádku, IPv4 FreeRADIUS se spojením do MySQL nám funguje.

7.4.2 Instalace pro IPv6

Protože celá tato práce je zaměřena na nasazení IPv6, i FreeRADIUS zprovozníme pod tímto protokolem. Nejdříve jsem zkusil standartní instalaci přenastavit tak, aby naslouchala na

všech IPv6 adresách. Je třeba podotknout, že stejně jako isc-dhcp server neumí jedna instance aplikace FreeRADIUS poslouchat současně na IPv4 a IPv6 adresách.

1. Upravíme konfiguraci a restartujeme démona (v debug módu).

Kód:

```
vim radiusd.conf
listen {
  type = auth
  #ipaddr = *
  ipv6addr = ::
}
listen {
  type = acct
  #ipaddr = *
  ipv6addr = ::
}
/etc/init.d/freeradius stop
freeradius -X
...
radiusd: #### Opening IP addresses and Ports ####
listen {
        type = "auth"
        ipv6addr = :: IPv6 address [::]
        port = 1812
/etc/freeradius/radiusd.conf[240]: Error binding to port for ::
   port 1812
```

Zde narazíme na zásadní problém. Instalace distribuovaná s Ubuntu 10.04 sice tvrdí, že podporuje IPv6 (a hodnoty jsou i v ukázkových konfiguračních souborech), ale tato verze pod IPv6 **nefunguje**! Po ne úplně krátkém hledání jsem objevil, že problém by měl být způsoben zapnutým flag *–with-udpfromto* při konfiguraci a následné kompilaci distribučního balíku. Balík jsem tedy překompiloval. Doporučuji balíky kompilovat mimo samotný systém, na stejném OS se stejnou architekturou. Vyhneme se tím zanesení cílového systému zbytečnými balíky nutnými pro kompilaci, které i po odinstalování můžou nechat nepořádek.

2. Překompilování distribučního balíku.

Kód:

```
sudo apt-get build-dep freeradius
apt-get source freeradius
cd freeradius-2.1.8+dfsg
vim debian/rules
--with-udpfromto --> --without-udpfromto
dpkg-buildpackage -j6
```

Přepínač *-j6* spustí 6 vláken pro kompilaci. (PC které sloužilo pro testování disponuje čtyřmi jádry). Kompilace ovšem skončila s chybou. Zde jsem již začínal být poněkud rozzlobený, protože jsem nechápal, jak může distribuční balík „nejít“ zkompilovat. Zkusil jsem ještě vrátit konfigurační parametr zpět, ale bez výsledku. Poté jsem zkusil ještě

stáhnout verzi přímo od FreeRADIUS, ale výsledek byl zcela shodný. Jako poslední možnost jsem zkusil ještě vypnout kompilaci ve více vláknech a k mému obrovskému údivu se balík zkompiloval. Vývojáři tedy zcela očividně zapomněli důsledně sledovat závislosti mezi balíky. Pro úspěšnou kompilaci je tedy třeba vynechat přepínač *-j6*. Vzniknou nám následující balíky (v závislosti na architektuře):

Kód:

```
freeradius_2.1.8+dfsg-1ubuntu1_i386.deb
freeradius-common_2.1.8+dfsg-1ubuntu1_all.deb
freeradius-dbg_2.1.8+dfsg-1ubuntu1_i386.deb
freeradius-dialupadmin_2.1.8+dfsg-1ubuntu1_all.deb
freeradius-iodbc_2.1.8+dfsg-1ubuntu1_i386.deb
freeradius-krb5_2.1.8+dfsg-1ubuntu1_i386.deb
freeradius-ldap_2.1.8+dfsg-1ubuntu1_i386.deb
freeradius-mysql_2.1.8+dfsg-1ubuntu1_i386.deb
freeradius-postgresql_2.1.8+dfsg-1ubuntu1_i386.deb
freeradius-utils_2.1.8+dfsg-1ubuntu1_i386.deb
libfreeradius2_2.1.8+dfsg-1ubuntu1_i386.deb
libfreeradius-dev_2.1.8+dfsg-1ubuntu1_i386.deb
```

3. A instalaci potřebných balíků provedeme následovně:

Kód:

```
dpkg -i freeradius_2.1.8+dfsg-1ubuntu1_i386.deb \
freeradius-common_2.1.8+dfsg-1ubuntu1_all.deb \
freeradius-mysql_2.1.8+dfsg-1ubuntu1_i386.deb \
freeradius-utils_2.1.8+dfsg-1ubuntu1_i386.deb \
libfreeradius2_2.1.8+dfsg-1ubuntu1_i386.deb
```

4. Vyzkoušíme znovu spustit server a nyní již běží

Kód:

```
/etc/init.d/freeradius stop
freeradius -X
...
radiusd: #### Opening IP addresses and Ports ####
listen {
        type = "auth"
        ipv6addr = :: IPv6 address [::]
        port = 1812
}
listen {
        type = "acct"
        ipv6addr = :: IPv6 address [::]
        port = 1813
}
Listening on authentication address :: port 1812
Listening on accounting address :: port 1813
Listening on proxy address :: port 1814
Ready to process requests.
```

5. Vyzkoušíme tedy znovu spojení na *localhost*, tentokrát po IPv6. Překompilováním balíků jsme přidali podporu IPv6 i do aplikace *radclient*, bohužel nikoliv do aplikace *radtest*. Vyzkoušíme tedy spojení přímo přes *radclient* s přepínačem pro *-6* pro IPv6.

Kód:

```
echo "User-Name=sqlmd5,Password=secr3t" | radclient -6 localhost
   auth testing123 -x
Sending Access-Request of id 192 to ::1 port 1812
        User-Name = "sqlmd5"
        Password = "secr3t"
radclient: no response from server for ID 192 socket 3
```

Zdá se, že server neodpovídá, ale v debug konzoli serveru je vidět, že nemáme zavedeného klienta pro IPv6.

Kód:

```
Ignoring request to authentication address :: port 1812 from
   unknown client ::1 port 43493
```

6. Přidáme tedy klienta do databáze a restartujeme FreeRADIUS server.

Kód:

```
INSERT INTO nas
  (id, nasname, shortname, type, ports, secret, community,
     description)
  VALUES
  (NULL, '::1', '::1', 'other', NULL, 'testing123', NULL, 'RADIUS
      Client');
```

7. A znovu vyzkoušíme

Kód:

```
echo "User-Name=sqlmd5,Password=secr3t" | radclient -6 localhost
   auth testing123 -x
Sending Access-Request of id 147 to ::1 port 1812
        User-Name = "sqlmd5"
        Password = "secr3t"
rad_recv: Access-Accept packet from host ::1 port 1812, id=147,
   length=20
```

A nyní je již vše v pořádku.

8. Vyzkoušíme ještě z jiného klienta (zde musíme také nainstalovat upravené balíky a přidat IP do databáze)
Server:

Kód:

```
INSERT INTO nas
  (id, nasname, shortname, type, ports, secret, community,
     description)
  VALUES
  (NULL, '2001:5c0:1515:b900::1', '2001:5c0:1515:b900::1', 'other
     ', NULL, 'testing123', NULL, 'RADIUS Client');
```

Klient:

Kód:

```
aptitude install freeradius-utils
dpkg -i freeradius-common_2.1.8+dfsg-1ubuntu1_all.deb \
freeradius-utils_2.1.8+dfsg-1ubuntu1_i386.deb \
libfreeradius2_2.1.8+dfsg-1ubuntu1_i386.deb
```

A provedeme test, zda byl problém odstraněn.

Kód:

```
echo "User-Name=sqlmd5,Password=secr3t" | radclient -6 2001:5c0
   :1515:b900::1 auth testing123 -x
Sending Access-Request of id 154 to 2001:5c0:1515:b900::1 port
   1812
        User-Name = "sqlmd5"
        Password = "secr3t"
rad_recv: Access-Accept packet from host 2001:5c0:1515:b900::1
   port 1812, id=154, length=20
```

Vše v pořádku.

9. Máme tedy plně funkční aplikaci FreeRADIUS komunikující s MySQL databází a pracující na IPv6.

7.5 Úpravy pro aplikaci

Databáze

Aby byl systém co nejvíce konzistentní, přidal jsem tabulky nutné pro AAA proces do databáze *IPv6*, kterou používá zbytek služeb. Úpravy byly následující:

- Překopírování tabulek z databáze *radius* do *IPv6* s vyjímkou tabulky *nas*
- Přejmenování tabulky *radcheck* na *User*
- Připojení do databáze

Úprava připojení do databáze:

```
# vim /etc/freeradius/sql.conf
server = "localhost"
login = "IPv6"
password = "<HESLO>"
radius_db = "IPv6"
nas_table = IPAddress
radcheck -> User
```

- Změna SQL dotazů generujících seznam klientských stanic

Původní dotaz:

```
nas_query = "SELECT id, nasname, shortname, type, secret FROM ${
    nas_table}"
```

Na nové

Nový dotaz:

```
nas_query = "SELECT idIPAddress id, inet6_ntop(IPAddress) nasname
    , inet6_ntop(IPAddress) shortname, 'other' type, 'testing123'
    secret FROM ${nas_table}"
```

Oddělení IPv4 a IPv6 konfigurací

Jak již bylo zmíněno výše, RADIUS postrádá schopnost naslouchat v jedné instanci současně na IPv4 a IPv6 sítích. Je ovšem možné spustit dvě instance, přičemž každá naslouchá na jiné verzi IP. Přestože změna v konfiuraci je pouze v jednom souboru, aplikace umožňuje předat jako parametr pouze celý adresář s konfigurací. Aby konfigurace zůstala kompaktní, vytvořil jsem dva konfigurační adresáře a původní obsah do nich nalinkoval. Změny provedené v původním adresáři se tedy projeví v obou instancích.

Kód:

```
/# cd /etc
/etc# mkdir freeradius4 freeradius6
/etc# ln -s freeradius/* freeradius4/
/etc# ln -s freeradius/* freeradius6/
/etc# cd freeradius4
/etc/freeradius4# rm radiusd.conf
/etc/freeradius4# ln -s radiusd4.conf radiusd.conf
/etc# cd ../freeradius6
/etc/freeradius4# rm radiusd.conf
/etc/freeradius4# ln -s radiusd6.conf radiusd.conf
```

A nyní lze jednotlivé instance spouštět jako

Kód:

```
freeradius -d /etc/freeradius4
freeradius -d /etc/freeradius6
```

8 Směrování

Kdykoliv potřebujeme, aby se paket mohl dostat z jedné sítě do druhé, potřebujeme jej směrovat. Máme možnost směrování nastavit na všech zůčastněných směrovačích ručně, nebo můžeme použít nějaký směrovací protokol. Pro potřeby navrhované sítě jsem se rozhodl použít OSPF[25] kvůli několika výhodám:

- Protokol je hojně rozšířený
- Podporuje IPv4 i IPv6
- Nalezneme jeho implementaci pro Cisco IOS i pro Linux
- Vyhovuje nárokům na rozsah sítě
- Velmi rychle konverguje

8.1 Zebra/Quagga

V případě mé práce jsem využil implementaci jménem Zebra[24]. Jedná se o implementaci podpory celého baíku směrovacích protokolů a konfigurace je velice podobná Cisco IOS. Pro zjednodušení instalace jsem využil odnož jménem Quagga[23], která se více zaměřuje na komunitní přístup než centraliozovaná Zebra. Proto je také standartně v nabídce balíků Ubuntu. Protože jsou mezi těmito implementacemi jenom minimální rozdíly, budu v dalším textu používat souhrné označení *zebra*. Instalaci provedeme následovně

Kód:

```
aptitude install quagga
```

Abychom mohli začít jednotlivé služby používat, nejdříve je musíme povolit v konfiguračním souboru

/etc/quagga/daemons:

```
zebra=yes
ospfd=yes
ospf6d=yes
```

Pro spuštění služby musí existovat příslušný konfigurační soubor. Ten může být například i prázdný. Následuje konfigurace hlavního daemona pro software Zebra. Zde je možné nastavit, která rozhraní budou použita a jejich adresy. Následující soubor slouží pouze jako ukázka.

/etc/quagga/zebra.conf:

```
hostname Router
password zebra
enable password zebra

interface eth0
description Interface to External Network
ip address 1.2.3.4/24
ipv6 address 2001:1::1/64

interface eth1
description Interface to Internal Network
ip address 10.0.0.1/24
ipv6 address 2001:A::1/64

! priklad staticke cesty
!ip route x.x.x.x/m y.y.y.y
```

Snažil jsem se konfiguraci co nejvíce zjednodušit, proto jsem se rozhodl redistribuovat pomocí OSPF statické cesty i přímo připojené sítě. Následuje jednoduchý příklad, jak by mohl vypadat konfigurační soubor k nastaveí OSPF pro IPv4.

/etc/quagga/ospfd.conf:

```
hostname Router
password zebra
enable password zebra

router ospf
 ! Nasledujici id musi byt jedinecne minimalne v ramci area
 router-id 0.0.0.1
 redistribute kernel
 redistribute connected
 redistribute static
 network 1.2.3.0/24 area 0.0.0.1
```

Dále potom jednoduchý konfigurační soubor pro OSPF pro IPv4.

/etc/quagga/ospf6d.conf:

```
hostname Router
password zebra
enable password zebra

router ospf6
 ! Nasledujici id musi byt jedinecne minimalne v ramci area
 router-id 0.0.0.1
 redistribute kernel
 redistribute connected
 redistribute static
 ! Pri redistribuci odfiltrujeme jine nez globalni adresy
 area 0.0.0.1 range 2001::/64
 ! Pouzijeme sw Zebra pro rozesilani router advertisment zprav
 no ipv6 nd suppress-ra
 ipv6 nd ra-interval 10
 interface eth0 area 0.0.0.1
 interface eth1 area 0.0.0.1
```

Pokud budou v sítí figurovat i přístroje s Cisco IOS, nastavíme je tak, aby komunikovali v rámci jedné area se software Zebra. Postup může být následující:

Kód:

```
ipv6 router ospf 0
router-id 0.0.0.2
interface fa 0/0
ipv6 ospf 0 area 0.0.0.1
```

Routovací protokol OSPF pro IPv4 i IPv6 jsem vyzkoušel v rámci několika virtuálních strojů s OS Ubuntu, kdy v případě IPv4 se redistribovala cesta k mému ISP, v případě IPv6 cesta do tunelu popsaného v kapitole 12.2 a systém plně fungoval. Funčnost pro platformu Cisco se mi nepodařilo otestovat. Zkoušel jsem sice vytvořit propojení s emulovaným IOS spuštěným pomocí aplikace GNS3, bohužel se mi nepodařilo navázat sousedskou vazbu mezi směrovači. Chybu připisuji příliš komplikovanému spojení (kdy nejdříve vytvořím virtuální rozhraní `tap0` pro spojení do GNS3 a poté most mezi `tap0` a virtuálním rozhraním `vmnet1`, do kterého je připojený OS hostující software Zebra). Jsem ovšem přesvědčený, že na reálném hardware by komunikace probíhala v pořádku.

9 Databáze

9.1 Návrhová omezení

V principu by mohla aplikace fungovat na dvou přístupech

1. Načíst hodnoty z existujících konfiguračních soubourů, ty poté upravit a znovu uložit. Výhodou tohoto přístupu je možnost konfigurační soubory ručně upravovat. Nevýhodou je pracné parsování. Takto napsaná aplikace také není příliš flexibilní co se týče změn ve struktuře souborů.

2. Všechny hodnoty držet v databázi a konfigurační soubory generovat. Výhoda je ve snadné centralizované správě, zálohovatelnosti a flexibilnosti systému. Také je zde možnost soubor ručně editovat, ale pouze pro dočasné účely nebo testování. Nevýhodou je větší náročnost na systém a nutnost počátečního zavádění hodnot do systému.

Zvolil jsem druhou možnost: Uskladnění všech dat v databázi. Aplikace se poté bude moci snadno rozšiřovat o další komponenty. Také představa správy velkého množství stanic bez centralizace je pro každého administrátora noční můrou.

9.2 Volba datových typů

Při volbě datových typů pro sloupce tabulek jsem musel vzít v úvahu několik důležitých aspektů spojených s optimalizací ukládání adres do databáze. Je třeba brát ohled na rozdílnou strukturu IPv4 a IPv6 adres. Zde jsou výhody a nevýhody jednotlivých možných přístupů.

- Oddělené tabulky IPv4 a IPv6
  + Snadnější indexace
  − Složitější práce s tabulkami z pohledu aplikace
  − Nepřehlednost

- Sloučené tabulky IPv4 a IPv6
  + Jednodušší práce s tabulkou z pohledu aplikace
  − Nároky na datový typ
  − Složitější vkládání a vyhledávání
  − Požadována schopnost rozlišit jednotlivé typy od sebe

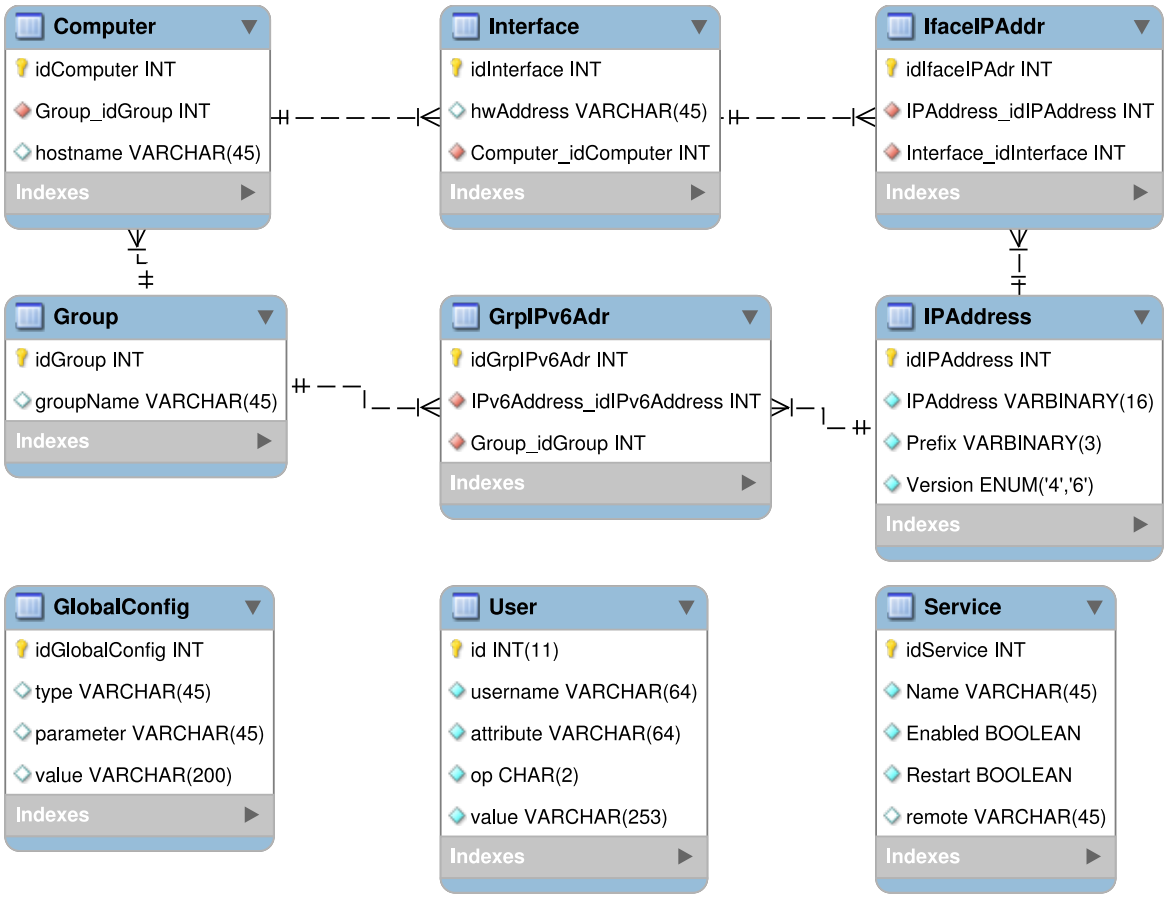
- Datový typ CHAR,VARCHAR
  + Čitelné záznamy při ruční editaci
  + Jednoduché ukládání adres jako řetezec
  − Neefektivní vyhledávání, nutnost indexace

- Datový typ BINARY,VARBINARY
  + Rychlé vyhledávání
  + Velká úspora velikosti (VARBINARY)
  + Lepší kontrola nad vkládanými záznamy
  + Strojově snadno zpracovatelné
  − Pro člověka nečitelné záznamy[4]
  − Nutnost převodů uvnitř aplikace

- Použití specializovaných MySQL funkcí pro práci s IPv4 i IPv6
  + Vyřešené ošetření vstupů
  + Optimalizované vkládání
  − Složitější instalace
  − Snížená přenositelnost databáze

Kvůli přehlednosti jsem zvolil možnost sloučit tabulky pro IPv4 a IPv6 adresy do jedné a pro uložení samotné adresy jsem vybral datový typ VARBINARY.

[4] Aplikace phpMyadmin umí přímo zobrazit BINARY jako text



Obrázek 7: *Schéma databáze*

Při rozhodování jak pracovat ukládat adresy jsem narazil na zajímavé řešení vývojové skupiny společnosti *Watchmouse*, zabývající se monitoringem dostupnosti webových stránek. Společnost nabízí pod GNU licencí knihovny [10] do MySQL, které přidávají funkce pro práci s IPv4 i IPv6 s jednotným rozhraním. Základní funkce jsou inet6_pton, která převádí IP adresy do VARBINARY(16) formátu, a inet6_ntop, která provádí přesně opačný proces. IPv6 adresy jsou automaticky ukládány v minimalizovaném tvaru, což vede ke snížení velikosti ukládaných dat bez ztráty na informační hodnotě. Odpadá tak dvojí implementace stejné funkčnosti na straně aplikace i webového rozhraní - k informacím z databáze bude uniformně přistupováno pomocí výše zmíněných funkcí. Obrovská výhoda tohoto systému je kompatibilita s běžně implementovanými funkcemi. Např. v Javě je možné VARBINARY položku jednoduše převést na objekt typu InetAddress.

Nevýhodou je, že zmíněné doplňující knihovny se musí zkompilovat a naimportovat do databáze na daném stroji, což snižuje jednoduchost instalace celého systému i přenositelnost databází např. pomocí dump zálohy.

Prefix pro danou adresu je do tabulky ukládán jako sloupec typu VARBINARY(3)

9.3 Návrh databázové struktury

Výsledné schéma databáze kterou bude systém používat je na obr. [7] na str. [33].

- Tabulka **Group**
  Představuje skupinu počítačů. Základní funkcí je přidělení adres a podsítí dané skupině. V případě potřeby je možné doimplementovat i další parametry specifické pro danou skupinu.

- Tabulka **GrpIPv6Adr**
  Tabulka vzniklá rozkldem vazby N:N mezi tabulkami Group a IPAddress

- Tabulka **Computer**
  Představuje jeden počítač. Počítač může mít jedno nebo více rozhraní. Je charakterizován řetězcem hostname, který nemusí být unikatní[5].

- Tabulka **Interface**
  Představuje rozhraní počítače. Obsahuje kontrolu na jedinečnosti MAC adresy. K jednomu rozhraní může být přiřazeno více IP adres.

- Tabulka **IfaceIPAddr**
  Tabulka vzniklá rozkldem vazby N:N mezi tabulkami Interface a IPAddress

- Tabulka **IPAddress**
  Obsahuje jednotlivé IP adresy (IPv4 i IPv6), prefixy k adresám a verzi adresy.

- Tabulka **GlobalConfig**
  Tabulka obsahuje parametry konfigurace na globální úrovni pro všechny aplikace. Aplikace jsou rozlišeny dle typu. Dále tabulka obsahuje dvojici parametr : hodnota.

- Tabulka **User**
  Obsahuje definici uživatelských účtů. V současné době ji využívá pouze aplikace RADIUS, předpokádá se rozšíření do dalších aplikací.

- Tabulka **Service** V této tabulce jsou uloženy jednotlivé služby, které aplikace dovoluje konfigurovat společně s informací, na kterém zařízení určitá služba běží. Pomocí aplikace je poté možno tyto služby nepřímo povolit, zakázat a restartovat.

Pro návrh jsem použil software MySQLWorkbench[16], který umí přehledně pracovat se závislostmi v databázi a velmi komfortně umí synchronizovat model s existující databází.

9.4 Instalace MySQL a dodatečných knihoven

Nainstalujeme potřebné balíky a závislosti

Kód:

```
aptitude install -y mysql-server libc6-dev libmysqlclient-dev
    libidn11-dev
```

Protože chceme do MySQL importovat knihovny, které nejsou ve standartním nastavení serveru, je nutné upravit odpovídající AppArmor profil.

[5]Sice by unikátní být měl, ale v reálném životě se můžeme často setkat s duplicitou. Odstranění těchto problémů je na adminstrátorovi sítě, nikoliv na aplikaci

Kód:

```
/etc/apparmor.d/usr.sbin.mysqld : "/usr/lib/mysql/** mr"
```

A poté nainstalovat samotné knihovny

Kód:

```
wget https://bitbucket.org/watchmouse/mysql-udf-ipv6/get/tip.tar.
   bz2
tar xvfj tip.tar.bz2
cd watchmouse-mysql-udf-ipv6-tip
make
make install
service mysql reload
```

čímž se nahrají .so knihovny do `/usr/lib/mysql/plugins` Do databáze je pak třeba nahrát patřičné funkce, které využivají dané knihovny.

Kód:

```
CREATE FUNCTION inet6_ntop RETURNS STRING SONAME "mysql_udf_ipv6.so
   ";
CREATE FUNCTION inet6_pton RETURNS STRING SONAME "mysql_udf_ipv6.so
   ";
```

Funkce poté převádějí IPv4 a IPv6 adresy do VARBINARY formátu, tedy např.

Kód:

```
select inet6_ntop(inet6_pton('2001:4860:a005::68')),
inet6_ntop(inet6_pton('1.2.3.4'));
select length(inet6_pton('2001:4860:a005::68')),
length(inet6_pton('1.2.3.4'));
```

10 Perl

Nejdříve jsem jako jazyk pro napsání převážné části aplikace zvolil Perl. Jak píší sami autoři: Perl 5 je velice schopný programovací jazyk bohatý na funkce, který má za sebou více jak 23 let vývoje. Perl 5 je spustitelný na více než 100 platformách od přenosných zařízení po mainframy a je vhodný pro rychlé jednorázové scripty i pro velké projekty.[20]

Pro upřesnění se jedná o interpretovaný programovací jazyk.

Výhody programovacího jazyka Perl:

- Rozšířenost ve světě WWW formou CGI skriptů
- Nenáročnost na hostitelský systém (v porovnání např. s jazykem *Java*)
- Podpora objektového programování
- Velké množství rozšíření (CPAN moduly)
- Nativní podpora na OS 4.2

- Aplikace lze pomocí PHP spouštět přímo z prostředí apache což umožňuje sdílet knihovny a funkce.

Během programování jsem se snažil udržet na vhodné míře abstrakce, aby byl software dále rozšiřitelný. Také jsem chtěl využít třídy zpracovávající logiku aplikace opakovaně, tj. v systémové části i GUI. Bohužel jsem zjistil, že použití Perl modulů v php není ani zdaleka tak jednoduché, jak se zdálo. Zásadním problémem ovšem byla bezpečnost, která je U perl CGI často probíraným tématem. Rozhraní obsahují mnoho chyb a mezer typu exploit. Přešel jsem proto na programovací jazyk Java v kombinaci s knihovnami Hibernate a Java Server Faces. Toto řešení je sice těžší na počáteční implementaci, ovšem poté lze aplikace velice snadno rozšiřovat. Navíc pro práci je k dispozici mnoho volně dostupných nástrojů, které umožnují zaměřit se na samotnou logiku aplikace a přitom za programátora dělají spoustu činností.

11 Java

11.1 Úvodem

Java je objektově orientovaný programovací jazyk, který vyvinula firma Sun Microsystems a představila 23. května 1995.

Java je jedním z nejpoužívanějších programovacích jazyků na světě. Podle Tiobe indexu je Java nejpopulárnější programovací jazyk. Díky své přenositelnosti je používán pro programy, které mají pracovat na různých systémech počínaje čipovými kartami (platforma JavaCard), přes mobilní telefony a různá zabudovaná zařízení (platforma Java ME), aplikace pro desktopové počítače (platforma Java SE) až po rozsáhlé distribuované systémy pracující na řadě spolupracujících počítačů rozprostřené po celém světě (platforma Java EE). Tyto technologie se jako celek nazývají platforma Java. Dne 8. května 2007 Sun uvolnil zdrojové kódy Javy (cca 2,5 miliónů řádků kódu) a Java bude dále vyvíjena jako open source.

Existuje několik teorií o původu jména tohoto jazyka, jedna z nich mluví o inspiraci u slangového označení kávy.[26]

11.1.1 Základní vlastnosti

- **jednoduchý** – jeho syntaxe je zjednodušenou (a drobně upravenou) verzí syntaxe jazyka C a C++. Odpadla většina konstrukcí, které způsobovaly programátorům problémy a na druhou stranu přibyla řada užitečných rozšíření.

- **objektově orientovaný** – s výjimkou osmi primitivních datových typů jsou všechny ostatní datové typy objektové.

- **distribuovaný** – je navržen pro podporu aplikací v síti (podporuje různé úrovně síťového spojení, práce se vzdálenými soubory, umožňuje vytvářet distribuované klientské aplikace a servery.

- **interpretovaný** – místo skutečného strojového kódu se vytváří pouze tzv. mezikód (bajtkód). Tento formát je nezávislý na architektuře počítače nebo zařízení. Program pak může pracovat na libovolném počítači nebo zařízení, který má k dispozici interpret Javy, tzv. virtuální stroj Javy - Java Virtual Machine (JVM).

  V pozdějších verzích Javy nebyl mezikód přímo interpretován, ale před prvním svým provedením dynamicky zkompilován do strojového kódu daného počítače (tzv. just in

time compilation - JIT). Tato vlastnost zásadním způsobem zrychlila provádění programů v Javě ale výrazně zpomalila start programů.

V současnosti se převážně používají technologie zvané HotSpot compiler, které mezikód zpočátku interpretují a na základě statistik získaných z této interpretace později provedou překlad často používaných částí do strojového kódu včetně dalších dynamických optimalizací (jako je např. inlining krátkých metod atp.).

- **robustní** – je určen pro psaní vysoce spolehlivého softwaru – z tohoto důvodu neumožňuje některé programátorské konstrukce, které bývají častou příčinou chyb (např. správa paměti, příkaz goto, používání ukazatelů). Používá tzv. silnou typovou kontrolu – veškeré používané proměnné musí mít definovaný svůj datový typ.

- **generační správa paměti** – správa paměti je realizována pomocí automatického Garbage collectoru který automaticky vyhledává již nepoužívané části paměti a uvolňuje je pro další použití. To bylo v prvních verzích opět příčinou pomalejšího běhu programů. V posledních verzích běhových prostředí je díky novým algoritmům pro garbage collection a tzv. generační správě paměti (paměť je rozdělena na více částí, v každé se používá jiný algoritmus pro garbage collection a objekty jsou mezi těmito částmi přesunovány podle délky svého života) tento problém ze značné části eliminován.

- **bezpečný** – má vlastnosti, které chrání počítač v síťovém prostředí, na kterém je program zpracováván, před nebezpečnými operacemi nebo napadením vlastního operačního systému nepřátelským kódem.

- **nezávislý na architektuře** – vytvořená aplikace běží na libovolném operačním systému nebo libovolné architektuře. Ke spuštění programu je potřeba pouze to, aby byl na dané platformě instalován správný virtuální stroj. Podle konkrétní platformy se může přizpůsobit vzhled a chování aplikace.

- **přenositelný** – vedle zmíněné nezávislosti na architektuře je jazyk nezávislý i co se týká vlastností základních datových typů (je například explicitně určena vlastnost a velikost každého z primitivních datových typů). Přenositelností se však myslí pouze přenášení v rámci jedné platformy Javy (např. J2SE). Při přenášení mezi platformami Javy je třeba dát pozor na to, že platforma určená pro jednodušší zařízení nemusí podporovat všechny funkce dostupné na platformě pro složitější zařízení a kromě toho může definovat některé vlastní třídy doplňující nějakou speciální funkčnost nebo nahrazující třídy vyšší platformy, které jsou pro nižší platformu příliš komplikované.

- **výkonný** – přestože se jedná o jazyk interpretovaný, není ztráta výkonu významná, neboť překladače pracují v režimu „just-in-time“ a do strojového kódu se překládá jen ten kód, který je opravdu zapotřebí.

- **víceúlohový** – podporuje zpracování vícevláknových aplikací

- **dynamický** – Java byla navržena pro nasazení ve vyvíjejícím se prostředí. Knihovna může být dynamicky za chodu rozšiřována o nové třídy a funkce, a to jak z externích zdrojů, tak vlastním programem.

- **elegantní** – velice pěkně se v něm pracuje, je snadno čitelný (např. i pro publikaci algoritmů), přímo vyžaduje ošetření výjimek a typovou kontrolu.[26]

11.2 Hibernate

Hibernate je knihovna pro objektově-relační mapování (object-relational mapping (ORM)) v jazyce Java. Poskytuje framework pro mapování objektově orientovaného doménového modelu na tradiční relační databázi. Snaží se vyřešit problémy s *Object-relational impedance mismatch* 11.2.5 nahrazením přímého persistentního přístupu do databáze vysokoúrovňovými funkcemi pro práci s objekty.

Základní funkcí Hibernate je mapování Java tříd na databázové tabulky (a také převod datových typů z jazyku Java na typy jazyka SQL). Hibernate také poskytuje nástroje pro datové dotazy a získání jejich výsledků. Hibernate generuje SQL volání a pokouší se oddělit vývojáře od ruční práce s návratovými množinami a převodu objektů. Tímto také umožňuje portablitu aplikace mezi všemi podporovanými SQL databázami za cenu pouze malého poklesu výkonu. (Kompatibilita je zajištěna pomocí tříd typu ovladač (*Driver*)).

11.2.1 Mapování

Mapování tříd z jazyka Java do databáze je prováděno pomocí konfigurace v XML souboru, nebo použitím speciálních anotací. (Pro účely programu jsem vybral mapování pomocí XML pro jeho přehlednost a snadnou upravitelnost). Hibernate dokáže použít existující konfiguraci (pomocí XML i pomocí anotací) k udržení aktuálnosti databázového schématu tzv. *Forward Engineering*.

Hibernate poskytuje nástroje pro uskutečnění 1:N (one-to-many) a N:M (many-to-many) relací mezi třídami. Navíc ke schopnosti spravovat asociace mezi objekty, dokáže Hibernate také spravovat reflexivní asociace, kdy má objekt 1:N relaci s ostatními objekty stejného typu.

11.2.2 Persistence

Perzistence ve světě Java je schopnost zachovat stav objektu mezi jednotlivými spuštěními aplikace (tedy např. uložením parametrů objektu do databáze). Hibernate poskytuje transparentní persistenci pro *Plain Old Java Objekty (POJOs)*[17]. Jediný závazný požadavek na perzistentní třídu je konstruktor bez jakýchkoliv argumentů (konstruktor může být i privátní). Doporučuje se také vlastní přetížení metod *equals()* a *hashcode()*, aby mohl být objekt korektně porovnán. (Tento přístup jsem použil také ve své práci).

Množiny dat načtené z databáze jsou typicky uložené v kolekcích jazyka Java, jako je *Set* nebo *List*. Podporovány jsou také generické typy (zavedené v Java 5). Těchto generických typů ve své práci hojně využívám. Hibernate může být nakonfigurován aby načítal objekty z databáze formou *Lazy Loading* (Tj. objekty jsou inicializovány až ve chvíli, kdy jsou potřeba, nikoliv při spuštění aplikace). Ve své práci jsem se rozhodl použít Lazy Loading, protože výrazně snižuje paměťové nároky na aplikaci za cenu jejího mírného zpomalení. V prostředí konfigurace nepředpokládám nutnost okamžité reakce, zatímco snížení paměťových nároků je jistě pro administrátora velkým přínosem. V souvislosti s funkcí Lazy Loading jsem musel vyřešit hlavně problémy s korektním otevíráním a uzavíráním spojení do databáze při čtení a zapisování změn. Pokud totiž např. uživatel upravuje objekt, jehož součástí je kolekce jiných objektů (které dosud nebyly načteny z databáze), musí být schopna aplikace po dokončení změn načíst dané objekty z databáze. V praxi to tedy znamená, že prakticky v každém okamžiku práce s aplikací musí být dostupné funkční spojení do databáze.

Relační objekty mohou být nakonfigurovány na kaskádování operací z jednoho na druhý. Např. objekt typu *Computer* může být nakonfigurován, aby kaskádoval své operace pro

přidání a odebrání *Interface*, takže jsou tyto operace zavolány i na všechny objekty typu *Interface* příslušející k danému počítači. Zde bych se rád pozastavil nad problémem na který jsem narazil. Při použití funkce *Reverse Engineering*(viz 11.2.6) jsem naivně věřil vygenerovanému kaskádovému mapování objektů z databáze. Bohužel konfigurace relací typu N:M (např. mezi *Interface* a *IpAddress*) byly vygenerovány s parametrem *cascade-all*, místo korektního *cascade-all-delete-orphans*. Tato chyba měla za následek, že v databázi zůstávaly objekty, na něž nevedly žádné reference. Chybu jsem se snažil „obejít“ získáním těchto osiřelých objektů ještě před odstraněním objektů, jež na ně ukazovaly a následným SQL dotazem na jejich odstranění. Řešení sice bylo funkční, ale zcela popíralo smysl Hibernate mapování. Nakonec jsem na chybu v typu kaskádování narazil zcela náhodou (při hledání jiné chyby spojené perzistencí objektů).

Další nástroj který Hibernate poskytje pro urychlení vykonávání SQL dotazů, je tzv. *Dirty checking*. Tento nástroj předchází zbytečným zápisům do databáze prováděním SQL update volání pouze na hodnoty, které byly před voláním upraveny.

11.2.3 ManagedBean

ManagedBean je speciální druh třídy typu Bean, která je registrována v JSF a je jím manažována. Od JSF v 2.0 je možné tyto Bean komponenty určovat pomocí anotací, což vedlo ke značnému zjednodušení správy. Hlavní výhodou ManagedBean je, že mohou být referencovány z jiných Bean pomocí volání ManagedProperty. Obě Bean komponenty musí být v odpovídajícím rozsahu (viz. dále).

Ukázka ManagedBean:

```
// Pojmenovani ManagedBean
@ManagedBean(name="mb")
public class BeanClass

// Zavolani ManagedBean
@ManagedProperty("#{mb}")
private BeanClass mb;
```

11.2.4 Rozsahy

Rozsah (anglicky Scope) je speciální tag importovaný z knihoven Hibernate. Určuje viditelnost instance třídy pro ostatní instance a také určuje, kdy daná instance zaniká. Řízení je nutné, aby v paměti nezůstávaly instance, které již nejsou potřeba. Také je na druhou stranu nutné Hibernate určit, která instance má přetrvat. Nasleduje popis jednotlivých druhů a jejich chování, včetně popisu, kde byly v aplikaci použity.

- **NoneScoped**
  Objekty, jejichž rozsah není explicitně určen nejsou viditelné v žádné JSF stránce. Pokud jsou použity v konfiguračním souboru, určují ManagedBeans, které jsou použity jinými ManagedBeans v rámci aplikace. Objekty, jejichž rozsah není určen mohou používat ostatní objekty, které nemají určen rozsah.
  Použití: Všechny třídy kromě tříd typu *Bean*

- **RequestScoped**

Objekty v tomto rozsahu jsou viditelné od začátku dotazu po jeho dokončení. Rozsah vzniká na začátku dotazu a končí v okamžiku, kdy je odpověď odeslána klientu. Pokud jsou požadavky přesměrovány, je objekt viditelný i v přesměrované stránce, protože stránka je zobrazena v rámci jednoho cyklu typu Požadavek-Odpověď. Objekty v tomto rozsahu mohou používat NoneScoped, RequestScoped, SessionScoped a ApplicationScoped objekty.
Použití: Pouze u třídy typu Bean řídící objekt Interface (instance není používana žádnou jinou třídou)

- **ViewScoped**
  Objekty v tomto rozsahu jsou znovu vytvořeny při každém dotazu z/do stejného pohledu. (Tj. při dotazech v rámci jedné stránky). Zůstávají platné dokud klient neopustí stránku nebo není ukončena session. Objektu v tomto rozsahu mohou používat NoneScoped, SessionScoped a ApplicationScoped objekty. **Použití:** Třídy typu Bean řídící objekty zajišťující generování konfiguračních souborů.

- **SessionScoped**
  Objekty v tomto rozsahu jsou viditelné v kterémkoliv cyklu typu Požadavek-Odpověď, který náleží k session. Objekty v tomto rozsahu mají perzistentní stav mezi jednotlivými dotazy a zůstávají platné v paměti, dokud nejsou explicitně označeny za neplatné, nebo není ukončena session. Session je většinou ukončena voláním *commit()*. Objekty v tomto rozsahu mohou používat ostatní NoneScoped, SessionScoped nebo ApplicationScoped objekty.
  Použití: Třídy typu Bean řídící Group a Computer objekty. Např. při výběru objektu typu Computer na stránce se správou skupin, je nastavena proměnná typu Computer v ManagedBean a následně je zavolána stránka se správou objektů typu Computer. Na této stránce je z příslušné ManagedProperty načtena proměnná zpět. Pokud by objekty zanikly po zaslaní dalšího požadavku, nebyla by na ně již možná reference. Proto musí být objekty v rámci jedné session. Pokud by objekty naopak byly ApplicationScoped, byly by jejich instance drženy v paměti i po ukončení session, přestože by na ně již nebyla žádná reference.

- **ApplicationScoped** Objekt v tomto rozsahu je viditelný všemi cykly typu Požadavek-Odpověď, všemi klienty používajícími aplikaci po dobu běhu aplikace. Objekty v tomto rozsahu mohou používat ostatní ApplicationScoped a NoneScoped objekty.
  Použití: Hlavní třída typu Bean řídící všechny ostatní Bean třídy.

Hibernate nabízí možnost spojit instance objektů po přechodu mezi jednotlivými session pomocí funkcí *merge(Object o)* a *update(Object o)*. V aplikaci nebyly tyto funkce záměrně použity, protože je možné, že k databázi bude přistupováno i přes jiné kanály než aplikaci samotnou. Tím by mohlo snadno dojít k nekonzistenci objektů a generovaní nepříjemných chyb. Proto jsou objekty po každém volání funkce *commit()* znovu načteny.

11.2.5 Object-relational impedance mismatch

Sada konceptualních a schématických problémů, které vznikají při použití relačního databázového systému (RDBMS) programem napsaným v objektově orientovaném jazyku. Obzvláště v situacích, kdy jsou objekty nebo třídy mapovány přímočaře na databázové tabulky nebo relační schemata.

Vyjmenujme nejčastěji vznikající problémy:

- **Enkapsulace** - Objektově orientované programy jsou navrženy tak, aby tvořily enkapsulované objekty, jejichž obsah je před okolím ukryt. Mapování takového privátního objektu do databázových tabulek může způsobit nestabilitu databází z pohledu OOP filozofie. Pro tvorbu privátního enkapsulovaného objektu v OOP je totiž nutné splnit mnohem mírnější podmínky než ve srovnání s databází, kde data musí být veřejně dostupná pro **změnu**, **kontrolu** a **dotazy**.

  RDBMS spíše používá protekční a bezpečnostní mechanismy na bázi pravidel (rule-based) a rolí (role-based), zatímco OOP používá kontrolní mechanismy na úrovni rozrhaní (interface). Vložením prvku do RDBMS tento prvek automaticky získává standartní množinu relačních a databázových operací, které nad ním mohou být prováděny. Restrikce operací jsou často prováděny inkrementálním odebíráním operací z této množiny dle potřeby. Např. objektům, které nemají mít schopnost upravovat jinou než svojí část databáze je odepřen přístup k tabulkám jiných objektů.

  Na druhé straně stojí enkapsulace, která zamezuje komunikaci objektu s okolním světem, dokud nejsou explicitně definována rozhraní, která tuto komunikaci zprostředkují. (Nicméně většina RDBMS nabízí možnost uložit vlastní procedury, které sdílejí některé charakteristiky OOP enkapsulace.

- **Přístup k datům** - V relačních modelech je rozdíl mezi privátními a veřejnými atributy definován spíše jako vlastnost, kterou by bylo vhodné splnit, zatímco u OOP je tento atribut u dat absolutní.

- **Rozhraní, třídy, dědičnost a polymorfismus** - Přístup k objektům v OOP je pokud možno nejlépe provádět pomocí rozhraní, které společně poskytují jediný přístup k vnitřním hodnotám objektu. Na druhou stranu relační model používá odvozené relační proměnné (pohledy, views) pro poskytutí rozdílných perspektiv a omezení za účelem zajištění integrity dat. OOP třídní koncepty jako je dědičnost a polymorfismus nejsou relačním modelem poskytnutelné.

- **Rozdíly mezi datovými typy** - Jeden z největších rozdílů mezi existujícími OO a relačními jazyky je v systému, jakým jsou definovány typy. Relační model striktně zakazuje předávání atributů jako reference (nebo ukazatele), zatímco OO jazyky preferují a očekávají předávání referencí. Skalární typy a jejich operační sémantika je také často více či méně rozdílná, což způsobuje probémy s mapováním.

  Například většina SQL systémů podporuje řetězce znaků s rozdílným kódováním a omezením na délku (typy s předem neznámou délkou mají negativní dopad na výkon). Oproti tomu většina OO jazyků bere kódování pouze jako argument při třídění a řětězce jsou dynamicky mapovány do dostupné paměti.

  Další podobný příklad je, že SQL systémy často při porovnávání ignorují prázdné místo na konci řetězce, zatímco OO knihovny pro práci s řetězci ne.

11.2.6 Reverse Engineering

V původním návrhu Hibernate byl stanoven předpoklad, že designér navrhne software a k němu příslušné databázové struktury budou vytvořeny následovně dle entit v tomto software.

V praxi se ovšem s tímto přístupem setkáme spíše vyjímečně. V naprosté většině je nejdříve navržen databázový model, ke kterému je poté vytvořen příslušný software. Tím vzniká problém, jak vnést již existující databázovou strukturu do programu se všemi závislostmi, cizími klíči, kaskádami apod. Zde přichází ke slovu Reverse Engineering.

Reverzní inženýrství (Reverse Engineering , RE) je označení pro proces, jehož cílem je odkrýt princip fungování zkoumaného předmětu (např. mechanického zařízení nebo počítačového programu), většinou za účelem sestrojení stejně či podobně fungujícího předmětu (nemusí však být výslovnou kopií originálu). Reverzní inženýrství může být v závislosti na situaci a právním systému nelegální (např. jako průmyslová špionáž nebo porušení duševního vlastnictví), ne však ve všech státech světa stejně.[26]

V případě Hibernate je možné RE použít pro dodatečné generování tříd dle tabulek v databázi a k nim odpovídající konfigurační soubory pro Hiberate[6]. Tuto činnost lze samozřejmě vykonat i manualně, ale ve většině případů se třídy obsahují převážně jednoduché metody typu *get* a *set*. Navíc je pravděpodobnější, že programátor udělá nějakou drobnou, těžko odhalitelnou chybu. Na druhou stranu je nutné vygenerované třídy a konfigurace zkontrolovat z důvodů, o nichž se zmíním dále.

Pro RE jsem využil vestavěnou funkci v NetBeans. Podrobný návod je možné nalézt na stránkách NetBeans[18]. Vygenerované konfigurace bylo třeba následovně upravit:

- Mapování typu N:M prováděná přes vazební tabulky nebyla vygenerována. Bylo tedy nutné dodat ručně např.:

  Kód:

  ```
  <set cascade="all-delete-orphan" name="IPAddresses" table="
      IfaceIPAddr">
    <key column="Interface_idInterface"/>
    <many-to-many class="dbEntities.Ipaddress" column="
        IPAddress_idIPAddress"/>
  </set>
  ```

- Generování obsahuje bug[9], který se projeví následovně:
  Pokud se v nějaké tabulce nachází sloupec, který se jmenuje „Version“, mapování bude vypadat následovně:

  Kód:

  ```
  <version name="version" type="string">
    <column name="Version" length="2" not-null="true" />
  </version>
  ```

  což je zcela chybně. Správné mapování vypadá následovně:

  Kód:

  ```
  <property name="version" type="string">
    <column length="2" name="Version" not-null="true"/>
  </property>
  ```

[6]Popř. anotace přímo ve třídě

Trvalo mi několik hodin, než jsem na příčinu problému přišel. Zajímavé je, že tento bug byl popsát už před pěti lety, ale stále nebyl vyřešen. Na stánkách Hibernate je bug označený jako vyřešený a uzavřený, ovšem zřejmě není.

- Další problém byl zcela způsoben nepozorností. Pokud jsem do databáze vkládal pomocí Hibernate pouze jednu IP adresu na jeden Interface, chyba se neprojevila. Pouze při vložení více IP adres již začal hibernate vracet chyby s porušením vazeb cizích klíčů. Nejříve jsem chybu hledal ve špatné posloupnosti volání metod *save* popřípadně *commit*. Problém se ovšem vyskytl i v případě, že jsem se pokusil nahrát již existující entitu Interface z databáze a k ní jsem se pokusil přidat další IP adresu. Když se problém objevil i v tomto případě, bylo jasné, že je buď chyba v mapování nebo v databázi. Mapování mezi tabulkami Interface a IPAddress je shodné jako pro vazbu mezi tabulkami Group a IPAddress, které fungovalo bez problémů. Poté tedy přišla na řadu databáze. Při vývoji databázové struktury v aplikaci MySQLWorkbench, jsem nedopatřením špatně označil cizí klíče v tabulce *IfaceIPAddr*. Místo správných:

Kód:

```
CONSTRAINT `fk_table1_IPv6Address1`
    FOREIGN KEY (`IPAddress_idIPAddress` )
    REFERENCES `IPv6`.`IPAddress` (`idIPAddress` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
CONSTRAINT `fk_table1_Interface1`
    FOREIGN KEY (`Interface_idInterface` )
    REFERENCES `IPv6`.`Interface` (`idInterface` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

jsem generoval chybně

Kód:

```
CONSTRAINT `fk_table1_IPv6Address1`
    FOREIGN KEY (`idIfaceIPAdr` )
    REFERENCES `IPv6`.`IPAddress` (`idIPAddress` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_table1_Interface1`
    FOREIGN KEY (`Interface_idInterface` )
    REFERENCES `IPv6`.`Interface` (`idInterface` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

Tj. cizí klíč *fk_table1_IPv6Address1* jsem chybně přebíral z tabulky *IfaceIPAdr* místo z *IPAddress*. Chyba byla viditelná dokonce v grafickém znázornění databáze (prázdným políčkem u cízího klíče *fk_table1_IPv6Address1*), bohužel jsem políčko přehlédl. Po opravení již ukládání fungovalo korektně. Velice zajímavá shoda náhod způsobila, že jsem tento problém neobjevil dříve. Většinou jsem totiž v databázi neměl vytvořenu více jednu entitu pro Group, Computer ani Interface. A protože všude v databázi je pro primární ID nastavena funkce "Auto Increment", shodovaly se ID s hodnotou 1. Proto

dosud fungovaly všechny dotazy nahrávající data z databáze.

- Pokud by měl být dodržen technologicky správný postup, nemělo by se již do vygenerovaných tříd nijak zasahovat, aby je bylo možné při případné změně databáze opětovně vygenerovat bez zásadních změn v aplikaci (tj. dodržet vrstvový model). Tento postup jsem si dovolil porušit na jednom místě, a to ve třídě odpovídající IP adrese. Zde jsem přidal několik metod pro získání a uložení IP adres a prefixů typu String. Vzhledem k dřívější volbě datového typu VARBINARY v databázi a řešení problému s Object-relational impedance mismatch11.2.5, jsou návratové hodnoty mapovány do typu *byte[ ]*. Protože tento typ jsem stejně kdykoliv po získání ihned převedl na String, bylo jednodušší jej převádět rovnou v třídě samotné. Proto je tedy typ použit jako parametr pro vytvoření instance třídy *InetAddress* a z ní lze snadno získat typ String. Následující příklad je úkázka, jak je možné postupovat.

  Kód:

```
...
byte[] address = Ipaddress.getIpaddress();
InetAddress inAddr = InetAddress.getByAddress(address);
String ip = inAddr.getHostAddress();
...
```

- **HelperClasses** Protože zasahování do samotných tříd vygenerovaných pomocí RE není doporučované, všechny metody potřebné pro práci s těmito třídami jsem se rozhodl umístit do pomocných tříd. V současnosti jsou určeny převážně pro snadné přidávání a odebírání hodnot do tříd hlavních. Je ovšem možné je snadno rozšířit o další funkce dle budoucích potřeb.

11.3 JSF

V základním slova smyslu je JavaServer Faces framework pro vytváření uživatelských rozhraní pro Webové aplikace. Jeho hlavní výhoda je, že zjednodušuje vývoj uživatelských rozhraní, která jsou velmi často obtížnou a zrádnou částí vývoje webových aplikací. Samozřejmě je možné vytvořit uživatelské rozrhaní za použití základních Java Web technologií (jako jsou Java servlety a JavaServer Pages) bez výkonného systému pro framework, který byl vytvořen pro nasazení ve velkých projektech. Tento přístup ovšem často vede ke spoustě problémům během vývoje a následné údržby software. JSF předcházejí tomuto problému tím, že nabízejí robustní framework se zavedenými vyvojářskými postupy, které byly převzaty z mnoha předchozích frameworků pro Web developement.

JavaServer Faces byly vytvořeny prostřednictvím Java Community Process (JCP) skupinou lídrů v těchto technologiích, včetně Sun Microsystems, Oracle, Borland, BEA a IBM společně s přispěním mnoha expertů se světa Javy a Webu. Projekt byl předáván mezi několika vedoucími, až se dostal do rukou Craigu McClanahanovi. Toto jméno Vám je možná povědomé, protože McClanahan je zakladatel oblíbené OpenSource Web aplikace Struts. Pod jeho vedením byla v roce 2004 vypuštěna první oficiální verze JSF.

JavaServer faces jsou vytvořeny pro zjednodušení vývoje uživatelských rozrhaní následujícími prostředky.

- Poskytuje developement zaměřený na kompontenty, přičemž nezávisí na klientovi[7].
- Zjednodušuje přístup k aplikačním datům a jejich management z uživtelského rozhraní.
- Automaticky spravuje uživatelské rozrhaní mezi jednotlivými dotazy více klientů jednoduchým a přesně definovaným systémem. (Viz. kapitola [11.2.3])
- Poskytuje vývojový framework, který je použitelný pro různorodé vývojáře s rozdílnými úrovněmi zkušností.

Mimo tyto přednosti má JSF ještě jeden velký přínos. Kombinuje totiž elementy, které byly nalezeny v průběhu mnoha let vývoje webových aplikací a kombinuje je do jednotného, výkonného a standartizovaného API. [4]

Proč tedy použít JSF

11.4 Generování konfigurace

Pro nastavení konfiguračních parametrů jednotlivých služeb, nutných k bezproblémové funkčnosti celé sítě, je zpravidla nezbytné vytvořit pro ně konfigurační soubory (o dalších metodách pojednává kapitola 2.2. Bylo tedy nutné, vytvořit příslušou programovou funkčnost, která by ze záznamů v databázi vytvořila příslušné konfigurační soubory. Většina stávajících Open-Source řešení funguje tak, že pro jednu konkrétní službu (např. DHCP) existuje jedna databáze, do které jsou ukládána všechna potřebná zařízení. Pro další služby (např. DNS) je pak nutné vytvořit novou databázi, což v naprosté většině vede k nekonzistenci dat. Pokud např. potřebujeme změnit IPv6 adresu pro stanici s názvem *pc1*, změníme její nastavění v DHCP. Pro správnou funkčnost DNS je ovšem nutné změnit i odpovídající záznam v DNS databázi. Celý tento problém jsem se snažil vyřešit navržením databáze obsahující zjednodušené entity z reálného světa . Tj. pracovní stanice, jejich skupiny (rozdělené např. podle geografického rozdělení), apod. Z konzistentích informací v této databázi poté čerpají služby své konfigurace.

Specifická nastavení pro jednotlivé služby jsou uchována v tabulce *GlobalConfig*.

Abych nepoužíval v implementaci opakující se části kódu sloužící k práci se soubory a dalším funkcím společným všem konfiguračním třídám, vytvořil jsem abstraktní třídu *Config*. Z té poté dědí jednotlivé třídy funkce a přidávají své vlastní. Základní schema je viditelné na obrázku 8 na str. 46

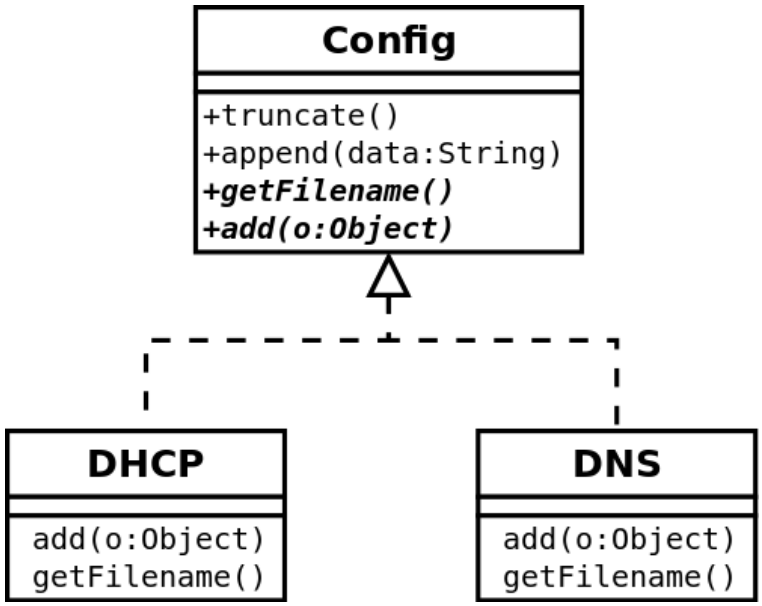
11.4.1 DHCP

Funkce třídy je vcelku jednoduchá. Hlavní je přetížená metoda *add(Object o)*, která dle typu objektu v parametru vykonává přidávání jednotlivých řádků do konfiguračního souboru. Soubory pro IP adresy verze 4 a 6 musí být oddělené. Základní syntaxe je celkem podobná. Jediný zásadní rozdíl je, že maska je u IPv4 zadávána decimálně, zatímco u IPv6 v CIDR notaci.

Isc-dhcp nepočítá v implemetaci s přiřazením více IP adres k jednomu hostname (ale je možné definovat více IP adres k jednomu identifikátoru rozhraní). Proto jsem problém vyřešil připojením suffixu s pořadovým číslem k hostname pro jednotlivé IP adresy.

V definici adres jsem postupoval následně:

[7] Bohužel úplná nezávislost na klientovi je asi pouze utopie



Obrázek 8: *Třídní diagram*

Zdroj	Parametr
Group	IP sítě + Prefix
Computer	Hostname
Interface	HWAddress
IPAddress	IP Zařízení

Pro převod masky z CIDR notace do decimální masky nutné pro uložení IPv4 konfiugurace jsem využil knihovnu *Commons Net 3.1 API*[14]

11.4.2 DNS

Funkce této třídy je velice podobná funkci třídy pro správu DHCP. Zde je ovšem generována celá následující struktura:

- Nová doména je přidána do konfiguračního souboru následujícím způsobem

/etc/bind/system/named.conf.local:

```
zone "domena.tld" {
type master;
file "/etc/bind/system/zones/master/domena.tld";
};
```

Syntaxe zónového souboru

Kód:

```
/etc/bind/system/zones/master/domena.tld
```

je pak již v souladu s kap. 6.4

11.4.3 RADIUS

Konfiguraci pro aplikaci FreeRADIUS není třeba generovat, protože seznam stanic a uživatelů si aplikace vytáhne sama z databáze, jak je popsáno v kapitole 7.5

11.5 Správa běžících služeb

Tato část aplikace pouze pracuje v databázi s tabulkou `Services`. Zde pomocí JSF rozhraní nastavíme které služby mají být spuštěny a na kterém zařízení. Je zde také možnost služby vzdáleně restartovat. O samotnou činnost se již postará jednoduchý skript v jazyce `bash`, který je spouštěn pravidelně pomocí služby `cron`. Tento skript se přihlásí na vzdálený server pomocí služby SSH s použitím privátního klíče a provede tam potřebné úpravy.

Na vzdáleném OS je jednorázově při zavádění naší aplikace provedena následující příprava. Příklad je proveden pro službu DHCP, ovšem postup pro ostatní služby je prakticky totožný. Počítače jsou označeny jako Zdroj **Z**, zde běží naše aplikace a výše zmíněny skript a Cíl **C**, kde běží služba DHCP.

1. **Z:** Přihlásíme se jako uživatel, pod kterým poběží server `tomcat`, v našem případě se jedná o uživatele `user`.

2. Vygenerujeme si dvojici veřejného a privátního klíče pro ssh. Poté si klíč zkopírujeme do schránky.

   Z:

   ```
   ssh-keygen
   cat ~/.ssh/id_rsa.pub
   ```

3. **C:** Vytvoříme dedikovaného uživatele pro správu služby. V našem případě se bude uživatel jmenovat `dhcp`

   C:

   ```
   useradd -m dhcp
   ```

   Přepínač `-m` způsobí vytvoření domovského adresáře.

4. **C:** Vytvoříme uživateli umístění pro uložení autorizovaných klíčů a vložíme do něj klíč získaný v bodě 2. Musíme také nastavit vhodná práva.

   C:

   ```
   mkdir -p /home/dhcp/.ssh/
   vim /home/dhcp/.ssh/authorized_keys
   chown dhcp:dhcp /home/dhcp/.ssh/ -R
   chmod 700 /home/dhcp/.ssh/
   chmod 600 /home/dhcp/.ssh/authorized_keys
   ```

   Nyní se již může uživatel `user` přihlásit jako uživatel `dhcp` bez použití hesla.

5. **C:** Vytvoříme si vhodnou adresářovou strukturu s příslušnými právy, kam budou nahrány konfigurační soubory a zkompilovaná aplikace pro DHCP.

C:

```
mkdir -p /opt/mic518/
chown dhcp:dhcp /opt/mic518/
chmod 700 /opt/mic518/
```

6. **Z:** Nahrajeme požadované zkompilované soubory.

Z:

```
cd /opt/mic518
tar c dhcp-4.2.3/server/dhcpd | ssh dhcp@<server> "cd /opt/mic518
    ; tar x"
tar c dhcp4.start.sh | ssh dhcp@<server> "cd /opt/mic518 ; tar x"
tar c dhcp6.start.sh | ssh dhcp@<server> "cd /opt/mic518 ; tar x"
```

7. **C:** Aby mohl uživatel `dhcp` spustit službu DHCP, potřebuje práva superuživatele. Zajistíme tedy, aby skripty nemohl editovat a vytvoříme mu tedy specifický záznam na konci souboru `/etc/sudoers`

C:

```
chown root:root /opt/mic518/dhcp4.control.sh /opt/mic518/dhcp6.
   control.sh
chmod 555 /opt/mic518/dhcp4.control.sh /opt/mic518/dhcp6.control.
   sh
visudo
dhcp ALL=(ALL) NOPASSWD: /opt/mic518/dhcp4.control.sh
dhcp ALL=(ALL) NOPASSWD: /opt/mic518/dhcp6.control.sh
```

Skript samotný poté pomocí ssh pouze nahraje vygenerovanou konfiguraci do požadovaného umístění a provede restart pomocí výše zmíněných privilegovaných skriptů.

Rozhodl jsem se takto oddělit aplikaci od služeb, protože z povahy webové aplikace vyplývá, že by měla sloužit hlavně pro konfiguraci hodnot a poté může být ukončena. Na druhou stranu nakonfigurované služby musí běžet neustále.

11.6 Správa uživatelských účtů

Správa uživatelských účtů se týká pouze uživatelů ze systému RADIUS. Je zde možné provádět následující základní operace:

- Vytvoření uživatele. Je zadáno uživatelské jméno a heslo, ze kterého je následně vygenerován MD5 hash a ten uložen do databáze.
- Odebrání uživatele.
- Změna hesla uživatele probíhá přes odebrání existujícího uživatele a jeho opětovné vytvoření.

11.7 Apache Tomcat

Apache Tomcat (nebo také zkráceně Tomcat, dříve známý jako Jakarta Tomcat) je open source webový server a kontejner pro servlety[8] vyvinutý Apache Software Foundation (ASF). Tomcat implementuje specifikace *Java Servlet* a *JavaServer Pages (JSP)* společnosti Oracle. Poskytuje prostředí napsané čistě v jazyce Java, ve kterém může běžet Java kód. Apache Tomcat obsahuje vlastní nástroje pro konfiguraci a management, ale může být také konfigurován ručně pomocí XML souborů.

Je nutné rozlišovat Apache Tomcat od Apache web serveru, což je webový server implementovaný v jazyce C. Tyto dva servery nejsou spolu svázány, ale často používány současně jako kompletní webově-aplikační balík.

Pro účely aplikace jsem vybral v současné době nejvyšší dostupnou stabilní verzi 7.0, která splňuje specifikace *Servlet 3.0* a *JavaServer Pages 2.2*.

Catalina je kontejner na servlety používaný aplikací Tomcat. Právě komponenta Catalina je zodpovědná za implementaci specifikací pro JSP. Dále má také na starosti databázi uživatelů a hesel a jejich role. Tyto uživatelé se často používají pro správu a monitoring serveru, ale mohou být použiti i pro autentizaci do samotných webových aplikací.

Instalace

Pro provoz aplikace jsem vybral verzi přímo od Apache Software Foundation, protože verze tomcat6 v Ubuntu repozitářích je velmi zastaralá. Ve Ubuntu verzi 11.10 by již měla být standartně nabízena verze tomcat7. Po vydaní nové LTS verze by tedy budoucí administrátor měl uvažovat o jejím nasazení.

Definice uživatelů se provádí v souboru *conf/tomcat-users.xml*. Pro základní management je nutné, aby uživatel měl přidělenou roli *manager-gui*

Kód:

```
<tomcat-users>
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <role rolename="manager-gui"/>
  <user username="tomcat" password="tomcat" roles="tomcat,role1,
     manager-gui"/>
</tomcat-users>
```

Instalace balíku ze stránek ASF je velice jednoduchá. Stačí balík rozbalit a spustit

Kód:

```
$ tar xzf apache-tomcat-7.*.tar.gz
$ cd apache-tomcat-7.0.26/bin
$./startup.sh
```

Velkou výhodou je, že pokud Tomcat spouštíme nad neprivilegovanými porty, nepotřebuje zvýšená oprávnění, což hodně zvyšuje celkovou bezpečnost systému. Server poskytuje velké množství nastavení, ale pro potřeby naší aplikace stačí nastavení standartní.

[8] Servlet je třída, použitá pro rozšíření schopností serveru. Uživatelské aplikace k těmto funkcím přistupují pomocí request-response programovacího modelu

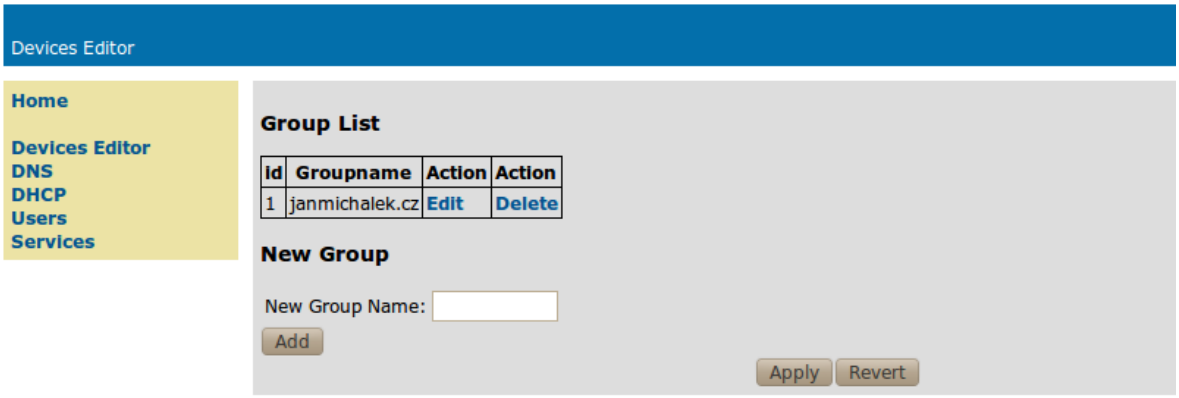
Samotný deployment aplikace spočívá ve zkopírování aplikačního balíku s příponou .war do Tomcat adresáře webapps.

12 Používání aplikace

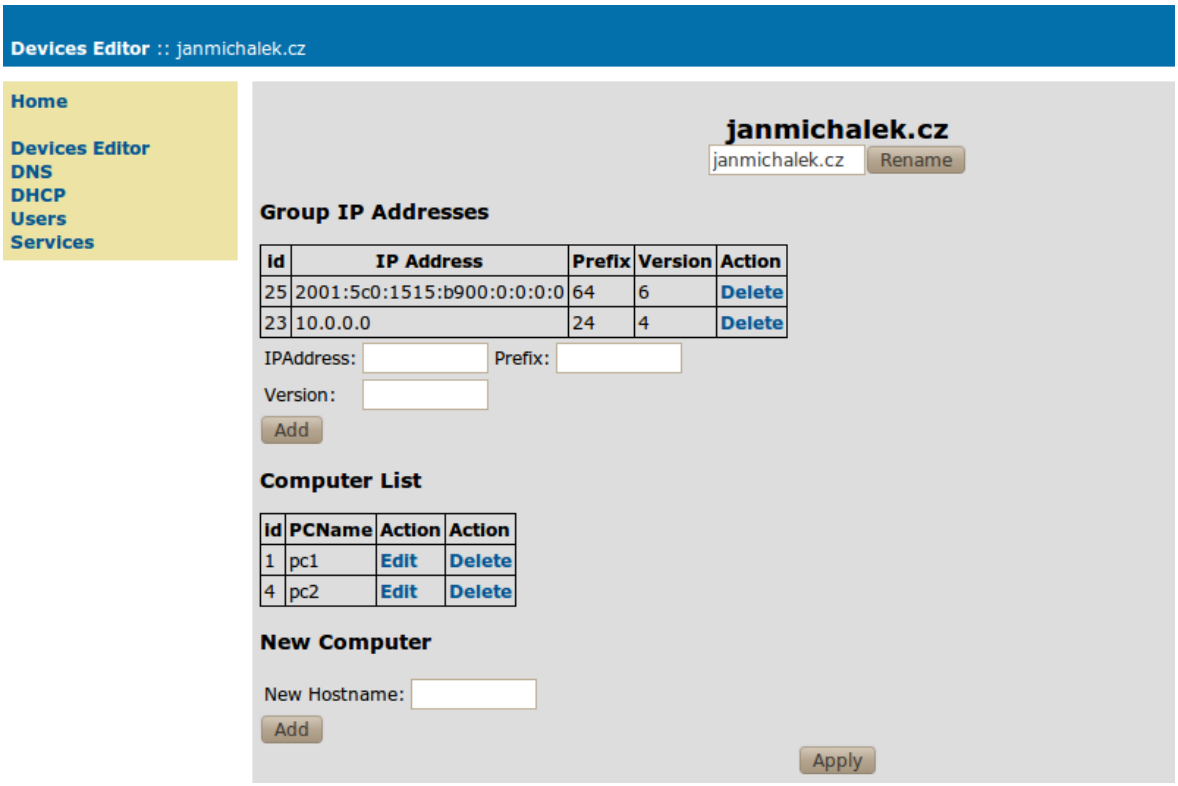
Samotné rozrhaní je anglickém jazyce, přičemž určitě není vyloučena možnost lokalizace. Vzhledem ke skutečnosti, že logika samotné aplikace je od rozrhaní oddělena, bude případný překlad s využitím služeb JSF jednoduchý na implementaci. Ve zbylé části této kapitoly bych chtěl nastínit jak rozrhaní vypadá a jak se používají jeho jednotlivé prvky, ale podrobný popis jednotlivých funkcí bude možné dohledat v dokumentaci.

12.1 Popis rozhraní

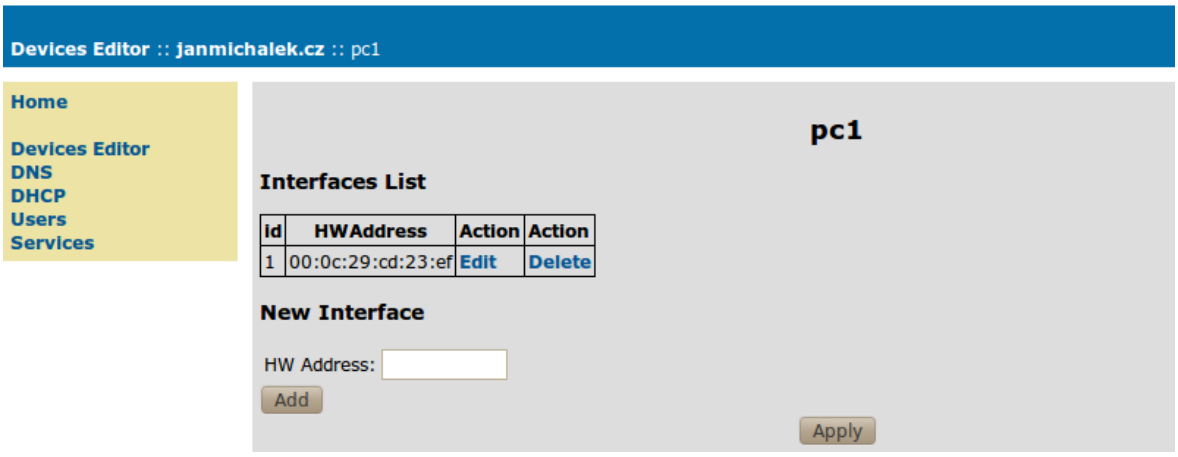
Umožňuje uživateli spravovat všechna zařízení v sítí. Protože systém pracuje hiearchicky, je na každé úrovni uživatelského rozhraní možné editovat příslušnou úroveň hiearchie.



Obrázek 9: *Základní obrazovka nejvyšší vrstvy - tedy vrstvy pro práci se skupinami zařízení je na obrázku. Do nižší vrstvy se dostaneme použitím tlačítka* `Edit` *u příslušné skupiny.*



Obrázek 10: *Rozhraní pro úpravu konkrétní vybrané skupiny - můžeme změnit IP adresy skupiny a spravovat zařízení, která do skupiny náleží.*



Obrázek 11: *Na obrázku je rozhraní pro úpravu zařízení (přidávání a odebírání síťových karet). V levé horní části jsou vidět tzv. breadcrumbs (chlebové drobečky). Podle nich se může uživatel zorientovat, kde se v hiearchii nachází. Také je může použít pro navigaci o libovolný počet úrovní výše.*

12.2 Testování aplikace

Na vývoji své práce jsem většinou pracoval v domácím prostředí, kde je připojení k Internetu realizováno pomocí ISP, který používá modem a jednu dynamickou veřejnou IP adresu. Proto jsem byl nucen toto připojení rozdělit pomocí routeru s funkcí NAT. Bohužel můj ISP neposkytuje možnost nativníího IPv6 připojení, proto jsem použil tunelování přes IPv4. Vyzkoušel jsem několik poskytovatelů tunnelBroker služby - *HurricaneElectrics* a *SixXS*, ale buď jsme narazil na problémy s registrací nového tunelu, nebo na velmi obtížné tunelování skrz NAT. Nakonec jsem použil řešení od *gogo6.net*, kde nabízejí klientskou aplikaci. Ta umí tunelovat přes UDP pakety. Tento tunnelBroker nabízí možnost použít jeho služby anonymně bez registrace a navíc v repozitářích Ubuntu se nachází i sestavený klient v balíčku gw6c.

Konfigurace klienta:

Hodnoty, které je třeba změnit pro správnou funkčnost gw6c klienta[9]:

/etc/gw6c/gw6c.conf:

```
host_type=router
prefixlen=56
if_prefix=eth1
userid=<USERID>
passwd=<PASSWD>
auth_method=any
log_stderr=1
log_file=1
server=amsterdam.freenet6.net
```

Abychom dostali pokaždé stejnou IPv6 adresu, je třeba se registrovat a pužívat pokaždé stejný server k připojení. Další výhodou registrace je větší prefix než `64`. Při tom vyskočí hlášení o chybejícím klíči pro server. Problém se dá obejít změnou v `/etc/defaults/gw6c`, kde přidáme řádek

/etc/defaults/gw6c:

```
CHECK_KEYFILE="no"
```

a poté restartujeme klienta.

Problémy se sdílením přiojení

gw6c aplikace sice používá skript pro automatizaci nastavení tak, aby se chovala jako router, ale v defaultním nastavení používá stateless konfiguraci, což je v rozporu s naším nastavením. Proto je nutné upravit template pro linux. Protože si nastavení routeru provádíme sami, zcela zakážeme sekci věnovanou pro router.

/usr/share/gw6c/template/linux.sh:

```
#if [ X"${TSP_HOST_TYPE}" = X"router" ]; then
if [ 1 -eq 2 ];then
```

[9] Za hodnoty userid a passwd uživatel doplní hodnoty uvedené při registraci

Dalé také zakomentujeme řádek, který při destrukci tunelu ruší lokální IPv6 adresu na zařízení uvedeném v konfiguraci (eth1).

/usr/share/gw6c/template/linux.sh:

```
#ExecNoCheck $ifconfig $TSP_HOME_INTERFACE inet6 del $TSP_PREFIX::1/64
```

A nyní je již klient schopen směrovat pakety z naší sítě do Internetu pomocí vytvořeného tunelu.

13 Závěr

Systém pro centralizovaný management IPv6 sítě hodnotím jako zdárně dokončený. Oproti původnímu zadání zaměřenému pouze na IPv6 jsem systém pojal jako nástroj pro celkovou správu sítě, tedy i pro stávající IPv4. Administrátor tak bude mít možnost spravovat celou topologii komfortně z jednoho umístění. Systém tedy splňuje všechny požadavky uvedené v zadání a ještě k tomu nabízí mnohé navíc.

Při výběru software pro správu jednotlivých služeb jsem se zaměřil co nejvíce na běžně používané nástroje, včetně dobře přenositelného jazyka Java pro napsání aplikace samotné. Je tak velká pravděpodobnost, že systém bude s malými úpravami provozuschopný i na jiném operačním systémů na platformě Linux, než je Ubuntu. Během vývoje samotného mi pak hodně pomohlo mnoho předmětů absolvovaných během studia. Další oporu jistě bylo současné zaměstnání ve společnosti, kde jsem se podílel na provozu služeb pro webhosting včetně práce s doménami a to vše současně nad protokoly IPv4 i IPv6.

Rozhodně pak doufám, že systém bude dále rozvíjen tak, aby obsahoval další moduly a neskončí založen a zapomenut. Příkladem rozšíření můžou být: Víceuživatelské rozhraní s různými úrovněmi přístupů - zde by jistě bylo vhodné zvážit využití existující struktury pro správu uživatelských účtů systémem RADIUS. Dalším rozšířením by mohly být statistiky a stav sítě, sledované pomocí protokolu SNMPv3. Chtěl jsem tuto funkci obsáhnout již v aktuální verzi aplikace, bohužel na její implementaci nezbyl čas. Jako další modul, který by jistě uvítalo mnoho administrátorů, bych doporučil správu IPv6 tunelů postavených přes IPv4 sítě. Poté bude možné opravdu komfortně testovat chování budoucí sítě s minimálním zásahem do stávající IPv4 struktury a to i v případě, že ISP neposkytuje nativní podporu IPv6. Každopádně navržený systém je velmi dobře použitelný v současném stavu.

Dále bych rád zmínil svůj osobní dojem ze současného stavu implementace IPv6 na platformě Linux, který je poněkud rozporuplný. Sice velké množství aplikací IPv6 oficiálně podporuje, ale přesto bylo třeba upravit jejich nastavení nebo aplikace překompilovat s jinými parametry, aby byla podpora IPv6 opravdu funkční. Tyto problémy můžou hodně administrátorů od nasazení IPv6 odradit. Na druhou stranu tato diplomová práce přináší ucelený přehled, jak se přes konkrétní problémy překlenout a výhody plynoucí z použití nového protokolu převyšují nad problémy spojené s jeho nasazením. Můžu tedy nasazení doporučit.

Zpracování práce bylo pro mě osobně velkým přínosem. Získal jsme mnohé nové znalosti o technologii IPv6, která je budoucností Internetu i LAN sítí a značně jsem rozšířil své znalosti služeb umožňujících chod Internetu. Zcela novým okruhem znalostí pro mě poté bylo sestavování aplikace pomocí vrstveného modelu, kdy na jednotlivých vrstvách jsou

- GUI sestavené pomocí JSF
- Aplikační logika
- Databázová logika napojená pomocí Hibernate

a odvažuji si tvrdit, že se mi podařilo kvalitně spojit mnoho technologií do jednoho konečného fungujícího celku.

Také myslím, že jsem plně využil znalosti nabyté v průběhu studia, ať se jedná o programování v jazyce Java nebo správa sítí. Navíc při očekávaném (i když neustále oddalovaném) přechodu na protokol IPv6 bude pro mě znalost tohoto protokolu na trhu práce výhodou.

Literatura

[1] Rooney, T.: *IP Address Management Principles and Practice*, ročník 16. Wiley-IEEE Press, 2011, ISBN 978-0470585870.

[2] Satrapa, P.: *Internetový protokol IPv6. 2. rozšířené vydání.* Praha: CZ.NIC, 2008, ISBN 978-80-904248-0-7.

[3] Hassel, J.: *RADIUS: Securing Public Access to Private Resources*. Publisher, 2002, ISBN 978-0-596-00322-7.

[4] Holmes, J.; Schalk, C.: *JavaServer Faces: the complete reference*. McGraw-Hill, Inc., 2006, ISBN 978-0071625098.

[5] Mirkovic, J.; Prier, G.; Reiher, P.: Attacking DDoS at the source. In *Network Protocols, 2002. Proceedings. 10th IEEE International Conference on*, IEEE, 2002, s. 312–321.

[6] isc-DHCP-server [online] 2008, [cit. 2012-04-28]. Dostupné z: http://www.isc.org/software/dhcp

[7] WIDE-DHCPv6 [online] 2008, [cit. 2012-04-28]. Dostupné z: http://wide-dhcpv6.sourceforge.net/

[8] Raggi, E.; Thomas, K.; Van Vugt, S.: *Beginning Ubuntu Linux: Natty Narwhal Edition*. Apress, 2011, ISBN 978-1430236269.

[9] HHH-3002 [online] 2007, [cit. 2012-04-28]. Dostupné z: https://hibernate.onjira.com/browse/HHH-3002

[10] Extending MySQL 5 with IPv6 functions [online] 2009, [cit. 2012-04-28]. Dostupné z: http://labs.watchmouse.com/2009/10/extending-mysql-5-with-ipv6%-functions/

[11] Sender Policy Framework [online] 2010, [cit. 2012-04-28]. Dostupné z: http://http://www.openspf.org2/

[12] dibbler-server [online] 2011, [cit. 2012-04-28]. Dostupné z: http://klub.com.pl/dhcpv6/

[13] Bind [online] 2012, [cit. 2012-04-28]. Dostupné z: http://www.isc.org/software/bind

[14] Commons Net 3.1 API [online] 2012, [cit. 2012-04-28]. Dostupné z: http://mvnrepository.com/artifact/commons-net/commons-net/3.1

[15] FreeRADIUS wiki [online] 2012, [cit. 2012-04-28]. Dostupné z: http://wiki.freeradius.org/

[16] Mysql Workbench [online] 2012, [cit. 2012-04-28]. Dostupné z: http://www.mysql.com/products/workbench/

[17] Plain Old Java Object [online] 2012, [cit. 2012-04-28]. Dostupné z: http://en.wikipedia.org/wiki/Plain_Old_Java_Object

[18] ReverseEngineering [online] 2012, [cit. 2012-04-28]. Dostupné z: http://wiki.netbeans.org/ReverseEngineering

[19] rhyshaden.com [online] 2012, [cit. 2012-04-28]. Dostupné z: http://www.rhyshaden.com/dns.htm

[20] Skriptovací jazyk Perl [online] 2012, [cit. 2012-04-28]. Dostupné z: http://www.perl.org/

[21] www.IPv6.cz [online] 2012, [cit. 2012-04-28]. Dostupné z: https://www.ipv6.cz/

[22] www.ubuntu.com [online] 2012, [cit. 2012-04-28]. Dostupné z: http://www.ubuntu.com

[23] Quagga [online] 2011, [cit. 2012-05-02]. Dostupné z: http://www.nongnu.org/quagga/

[24] Zebra [online] 2011, [cit. 2012-05-02]. Dostupné z: http://www.gnu.org/software/zebra/

[25] Open Shortest Path First [online] 2012, [cit. 2012-05-02]. Dostupné z: http://docwiki.cisco.com/wiki/Open_Shortest_Path_First

[26] Wikipedia.cz [online] [cit. 2012-04-28]. Dostupné z: http://wikipedia.cz

[27] Wikipedia.org [online] [cit. 2012-04-28]. Dostupné z: http://wikipedia.org